UNIVERSIDADE FEDERAL DE SANTA CATARINA
PROGRAMA DE PÓS-GRADUAÇÃO
EM CIÊNCIA DA COMPUTAÇÃO

Glaucio Adriano Fontana

ESTUDO E IMPLEMENTAÇÃO DE APRENDIZAGEM POR REFORÇO EM REDES NEURAIS PARA CONTROLE DE ROBÔS MÓVEIS

Mauro Roisenberg, Dr.
Orientador

Florianópolis, dezembro de 2003

Estudo e Implementação de Aprendizagem por Reforço em Redes Neurais para Controle de Robôs Móveis

Glaucio Adriano Fontana

Esta Dissertação foi julgada adequada para a obtenção do título de Mestre em Ciência da Computação, Área de Concentração Sistemas de Conhecimento e aprovada em sua forma final pelo Programa de Pós-Graduação em Ciência da Computação.

_____

Prof. Raul Sidnei Wazlawick
Coordenador do Curso

Banca Examinadora

_____

Prof. Mauro Roisenberg, Dr.
Orientador

_____

Prof. Jorge Muniz Barreto, Dr. Sc. A.

_____

Prof. João Bosco da Mota Alves, Dr.

_____

Prof. Edson Roberto De Pieri, Dr.

Para Dona Nair, um modelo de amor, força e honestidade, e por um grande presente da vida, minha mãe.

"Life has meaning only in the struggle. Triumph or defeat is in the hands of the gods...

...So let us celebrate the struggle!"

(Swahili Warrior Song)

"Só há sentido na vida com luta. O triunfo ou a derrota está nas mãos dos deuses...

...Então celebremos a luta!"

(Canção de Guerra Suaili)

Agradecimentos

A Deus.

Aos meus pais, Nair e Ary, por sempre acreditarem em mim e por terem me ensinado ao longo da vida o sentido das palavras trabalho e honestidade.

Aos meus irmãos Rosane, Mauro e Juciane, sobrinhos Fernanda, Fábio e Eduardo e cunhados Neida e Márcio pelo apoio incondicional.

Ao meu professor orientador Mauro Roisenberg, pela paciência, pelos conhecimentos passados, por toda a noção de boa escrita e por ser sempre tão humano e amigo.

A minha colega e amiga Luciene Marin, meu Xodozinho, pelas lições de amizade, perseverança, redes neurais e violão filosófico. Pela ajuda inestimável e fundamental à realização deste trabalho. E por ser sempre ela mesma, sem máscaras.

Ao meu tio Oreste Zorzo, pela mão sempre estendida e por ser a minha família em Santa Catarina.

Aos meus amigos de Santo Ângelo, Porto Alegre, Cascavel, (...) que mesmo longe, estiveram sempre perto, se fazendo presentes em e-mails, telefonemas, visitas, pensamento ou boas vibrações. O meu abração pra Sil, Dani, Sergião, Sica, Gringa, Rô, Sandra, Silvano, Lisiane, Vivi, Sandro, Simone Sasso, Simone Ramires, Nô, (...).

Aos meus amigos que fiz aqui: Michel, Themis, Elisa, Társis, Diogo, Jean Pierre, João, Andréa, Digu, Jimi, Gustavo, Rafael Rodrigues, (...), pela companhia agradabilíssima que sempre foram, pelas horas de bom papo e descontração tão necessárias. Aos amigos Rafael Rosa e Marcel pela solicitude, até mesmo em tempos de apagão (risos).

A todos os colegas de mestrado, os quais me ensinaram muito. Alguns deles, a conviver com idéias e personalidades tão diferentes das minhas.

Ao coordenador do curso de Informática da Universidade do Planalto Catarinense, Alexandre Perin de Souza, pelas faltas consentidas, necessárias ao término desta dissertação e aos alunos da sexta, sétima e oitava fases por compreenderem um professor muitas vezes cansado.

Enfim, a todos que de forma direta ou indireta contribuíram para a realização deste trabalho, a minha gratidão. Muito obrigado!

SUMÁRIO

LISTA DE FIGURAS

LISTA DE ABREVIATURAS

AA	Agente Autônomo
AR	Aprendizagem por Reforço
ART	Adaptive Resonance Theory
MLP	Multi-Layer Perceptron
NH	Neurônio Habilitador
RL	Reinforcement Learning
RNA	Rede Neural Artificial
TD	Temporal Difference
DL	Direção Leste
DN	Direção Norte
DO	Direção Oeste
DS	Direção Sul
L	Leste
N	Norte
O	Oeste
S	Sul

LISTA DE TABELAS

RESUMO

Robôs Móveis Inteligentes são sistemas computacionais que operam em ambientes dinâmicos e imprevisíveis. Eles interpretam dados obtidos pelos sensores que refletem eventos ocorridos no ambiente e executam comandos em motores que produzem efeitos no ambiente. O grau de autonomia de um agente está relacionado à capacidade de decidir por si só como relacionar os dados dos sensores com os comandos aos atuadores em seus esforços para atingir os objetivos para os quais foi projetado. Deste modo, a capacidade de aprendizado e adaptação do agente está intimamente relacionada com o seu grau de autonomia. Dentro do paradigma de inspiração biológica adotada na arquitetura de controle PiramidNet, Redes Neurais Artificiais são as ferramentas utilizadas para implementar a inteligência e o controle dos sistemas robóticos. Entretanto, mecanismos capazes de fazer o aprendizado permanente e em tempo de operação em sistemas robóticos controlados por Redes Neurais são escassos ou ainda estão em fase inicial de desenvolvimento. Neste trabalho, pretendeu-se estudar, propor e implementar métodos que possibilitassem o aprendizado em tempo real de Robôs Móveis Inteligentes controlados por Redes Neurais Artificiais. Para tal, propôs-se uma arquitetura neural de controle capaz de apresentar características de plasticidade e de estabilidade adequadas, utilizando redes ART – Adaptive Ressonance Theory e redes MLP – Multi-Layer Perceptron, associados a um esquema de aprendizagem por reforço como metodologia de aprendizado em tempo real.

ABSTRACT

Intelligent mobile robots are computational systems operating in unpredictable dynamic environments. They process data obtained from environment events capted by his sensors and execute commands that produce effects in the environment. The degree of an agent's autonomy is related to the capacity to decide by itself as to relate the data of the sensor ones with the commands to the effectors in its efforts to reach the objectives for which it was projected. This way, the learning capacity and the agent's adaptation is closely related with its autonomy degree. Inside of the paradigm of biological inspiration adopted in the PiramidNet control architecture, artificial neural networks are the tools used by implement robotic systems intelligence and control. However, mechanisms capable to do permanent on line learning in robotic systems controlled by artificial neural networks are scarse or they are still in initial phase of development. This master thesis intended studying, proposing and implementing methods that proceed on line learning in intelligent mobile robots controlled by artificial neural networks. A control neural architecture was proposed to perform plasticity-stability features, using ART (Adaptive Resonance Theory) neural network and MLP (Multi-Layer Perceptron) neural networks, associated to Reinforcement Learning concepts working as a on line learning methodology.

1 Introdução

1.1 Motivação

> "A inteligência do ser humano, aliada a sua mobilidade e criatividade, o levou a desenvolver mecanismos que , uma vez dotados de inteligência e mobilidade, viessem a substituí-lo nas tarefas repetitivas, estressantes ou perigosas. Estes mecanismos são conhecidos como Agentes Autônomos." (ROISENBERG 1996)

Outras definições em ROISENBERG(1996) são dadas para Agentes Autônomos: "Um agente autônomo é um sistema computadorizado capaz de extrair informações do seu ambiente e através de alguma capacidade cognitiva mapear as informações extraídas em ações que eventualmente podem afetar o ambiente de modo a alcançar os objetivos para o qual foi projetado". E "Agentes são sistemas computacionais que operam em ambientes dinâmicos e imprevisíveis. Eles interpretam dados obtidos pelos sensores que refletem eventos ocorridos no ambiente e executam comandos em motores que produzem efeitos no ambiente."

O grau de 'autonomia' de um agente está relacionado à capacidade de decidir por si só como relacionar os dados dos sensores com os comandos aos motores em seus esforços para atingir objetivos, satisfazer motivações." A motivação está intimamente relacionada com os estímulos sensoriais e é sempre orientada a um objetivo. A motivação está ligada também ao grau de satisfação decorrente da realização de uma tarefa. Se o grau de satisfação é alto, a motivação será também alta e por conseqüência o sistema terá um bom desempenho.

A inteligência computacional é uma das técnicas que mais se prestam para desenvolver aplicações dotadas de alto grau de inteligência e robustez e que operam em ambientes dinâmicos e hostis, como é o caso da robótica. No entanto, entre as várias abordagens atualmente englobadas pela inteligência computacional, nem todas se prestam para implementação de agentes inteligentes nestas áreas de aplicação, como é o caso, por exemplo, das técnicas da abordagem simbólica ou deliberativa. Por outro lado, uma nova filosofia de abordagem para implementação de agentes autônomos tem sido desenvolvida e apresentado resultados bastante animadores. Esta abordagem, fortemente inspirada na Natureza é conhecida como Robótica Baseada em Comportamentos (*Behavior-Based Robotics*) (MAES,1990).

Este paradigma biológico com forte teor comportamental contrapôs-se ao aparato deliberativo das técnicas simbólicas de inteligência artificial que prevaleciam na robótica desde os anos 70. A arquitetura reativa *Subsumption* (BROOKS, 1986) (BROOK, 1991) foi uma das primeiras implementações da Robótica Baseada em Comportamentos. Brooks argumenta que o mundo não é um lugar pacífico e imutável e as suas representações são de pouco valor para os que necessitam sobreviver e tomar decisões imediatas. Comportamentos são as unidades elementares e camadas especializadas estão superpostas hierarquicamente. Os resultados surpreendentes alteraram os rumos da robótica. As variações de arquiteturas posteriores modificam a estratégia de coordenação, o nível de granularidade na definição dos comportamentos ou a codificação das respostas (conjuntos pré-especificados ou contínuas) (OLIVEIRA, 2001) (ARKIN, 1998).

Se a Natureza é fonte de inspiração, então é natural que as Redes Neurais Artificiais sejam as ferramentas utilizadas para implementar a inteligência e o controle dos sistemas robóticos, uma vez que elas são o correspondente modelo computacional do sistema nervoso, que dá inteligência e determina o comportamento dos seres vivos.

O processo de aprendizagem ocupa papel fundamental neste processo. Segundo (KOVACS, 1996): “Aquele que for mais eficiente neste processo sobreviverá melhor, terá

mais descendentes(...). Uma característica fundamental do sistema nervoso é sua inexorável e contínua modificação com a experiência(...). Aprender implica em armazenar, de alguma forma, informações passadas relevantes e utiliza-las em momento oportuno".

Assim sendo, dada a relevância e o desejo de se ter agentes que se adaptem ao mundo com o qual estão interagindo, com alto nível de autonomia e com capacidade para se ajustar positiva ou negativamente a estados de satisfação previamente estabelecidos e aprender com suas experiências, e com o intuito de se ater a agentes robóticos como objeto de estudo, este trabalho se propõe a estudar um dos métodos mais utilizados para adaptar sistemas de controle robótico conhecido como aprendizagem por reforço.

A aprendizagem por reforço, como o próprio nome já diz, vai reforçar um determinado comportamento causado através de recompensas ou punições. Recompensas aplicadas imediatamente após a ocorrência de uma resposta aumentam a probabilidade da recorrência desta ação, assim como prover uma punição depois da resposta vai decrementar a probabilidade dessa ação ser novamente feita. A aprendizagem por reforço (AR) ou *reinforcement learning (RL)* se refere a um tipo de aprendizagem por tentativa e erro, onde recompensas ou punições são usadas para alterar valores numéricos em um controlador.

WHITEHEAD&LIN(1996) citam algumas características importantes e/ou interessantes acerca de aprendizagem por reforço:

* AR é incremental e pode ser usada on-line (não necessita de uma fase explícita de treinamento);

* AR pode ser usada para aprender mapeamentos sensório-motores diretos, sendo apropriada para tarefas altamente reativas em que o agente deve responder rapidamente a eventos inesperados no ambiente;

* AR é válida em ambientes não-determinísticos;

* As arquiteturas de AR são extensíveis. Existem aplicações de AR que incorporam aspectos de planejamento, exploração inteligente, aprendizagem supervisionada e controle hierárquico.

Apesar de Redes Neurais Artificiais serem uma ferramenta bastante promissora para implementações de Robótica Baseada em Comportamentos, na maioria dos casos a aprendizagem é feita anteriormente à colocação do agente no ambiente. Mecanismos capazes de fazer a aprendizagem permanente e em tempo de operação em sistemas robóticos controlados por Redes Neurais são escassos ou ainda estão em fase inicial de desenvolvimento (RING, 1994) (MEEDEN, 1995) (DIETTERICH, 1997).

Este trabalho propõe a utilização de redes neurais artificiais promovendo a um agente aprendizagem por reforço de um comportamento como alternativa de aprendizagem em tempo real.

1.2 Justificativa

A aprendizagem por reforço como abordagem inteligente aparece sob muitas formas procurando resolver diferentes tipos de problemas sob vários prismas, como programação neurodinâmica e processos de decisão markovianos[1], por exemplo.

A aplicação de aprendizagem por reforço através de redes neurais artificiais é o foco de estudo desta dissertação, numa tentativa não-convencional de aplicar esta técnica, procurando prover autonomia e certo grau de inteligência ao objeto de estudo: robôs móveis com a intenção de que estes agentes aprendam com suas próprias experiências e em tempo real.

---

[1] A aprendizagem por reforço através dos tipos citados pode ser vista em (HAYKIN 2001) , (SUTTON&BARTO 1998) e (KAELBLING et al 1996)

A implementação traz redes neurais diretas (*multi layer perceptron*), que apresentam facilidade de implementação da técnica de aprendizagem por reforço via redes neurais artificiais, conectadas a redes neurais ART (*Adaptive Resonance Theory*), que são capazes de armazenar novos padrões sem "esquecer" os já reconhecidos anteriormente. Assim, tratando as situações que o robô móvel enfrentará no seu ambiente como padrões, como por exemplo nesse caso a disposição dos obstáculos, é possível resolver o problema de "evitar-colisão".

1.3 Objetivos

1.3.1 Objetivo Principal

Pretende-se, como objetivo principal deste trabalho, implementar um agente inteligente autônomo e adaptativo (capaz de sentir e reagir às mudanças do ambiente e aprender com sua experiência) capaz de aprender um comportamento através do estudo de arquiteturas de redes neurais e do método de aprendizagem por reforço como proposta de aprendizagem em tempo real.

1.3.2 Objetivos Específicos

Para alcançar o objetivo principal, verificam-se ainda outros objetivos, a alcançar:

- Estudo aprofundado da metodologia de aprendizagem por reforço, tipos de algoritmos e verificação de problemas;

- Estudo e implementação da aprendizagem de comportamentos em tempo real através de redes neurais ART, bem como associação destas redes com redes diretas com reforço, propondo novas arquiteturas;

- Demonstrar a emergência da inteligência do agente através da descrição de um comportamento por meio da implementação de redes neurais com aprendizagem em tempo real.

1.4 Organização

A organização final deste trabalho prevê sua organização em seis capítulos, como descritos a seguir:

O capítulo um descreve a introdução do tema proposto no trabalho através da motivação e da explicitação dos objetivos principais e secundários, ou seja, os que darão suporte aos primeiros.

O capítulo dois disserta sobre sistemas robóticos inteligentes baseados em comportamento. Isto é, discute o que será o objeto da aplicação da aprendizagem por reforço, a luz do paradigma "baseado em comportamento". Para tanto, o capítulo trata de noções da abordagem de controle de comportamento em robôs.

O capítulo três faz uma revisão bibliográfica sobre redes neurais artificiais, o instrumento escolhido para proporcionar a implementação da aprendizagem por reforço. Fazem parte do capítulo além das características gerais dessa metodologia uma seção sobre as redes neurais ART, tipo específico de arquitetura a ser estudado na implementação e fala-se também sobre o dilema plasticidade-elasticidade, que as redes ART vêm resolver.

O capítulo quarto aparece como uma fundamentação teórica do *Reinforcement Learning* ou aprendizagem por reforço. Os componentes, problemas e alguns algoritmos são explicitados no capítulo além da caracterização do que vem a ser, de fato, a técnica central de estudo deste trabalho.

O capítulo cinco apresenta a implementação da aprendizagem por reforço em sistemas robóticos baseados em comportamento, justificando de forma prática e progressiva a proposta de utilização de uma arquitetura que conecta redes neurais ART e diretas. As primeiras com o intuito de prover memória de ações aprendidas e as últimas simulando reforço na aprendizagem das situações de evitar colisão que o robô enfrentará.

O capítulo seis espelha as conclusões e perspectivas de trabalhos futuros acerca da utilização de redes neurais para a aprendizagem de agentes robóticos autônomos geradas por este trabalho de pesquisa e implementação.

2 Agentes autônomos e inteligência

O termo agente, segundo MURPHY (2000) é um conceito em inteligência artificial que permite a pesquisadores discutir as propriedades da inteligência sem se preocupar como esta inteligência é obtida deste agente em particular. O agente, conforme MURPHY (2000) é independente, possui seu próprio "cérebro", e pode interagir com o mundo para realizar mudanças ou mesmo sentir o que está acontecendo.

Agentes Autônomos (AA) são sistemas que interagem em ambientes dinâmicos, impossíveis de predizer, no qual tentam satisfazer um conjunto de metas. Agentes são adaptáveis se eles melhoram as suas competências em lidar com estas metas baseado na sua experiência.

O objeto de descrição de um AA é seu comportamento, como este agente transforma as informações captadas do ambiente por seus sensores em comandos para seus atuadores, fazendo com que se movimente e aja nesse mesmo ambiente.

À luz dos conceitos estudados, este trabalho utiliza os conceitos de agente e de robô como tendo igual teor, ou ainda como sendo sinônimos.

2.1 Robôs Móveis Inteligentes

Conceitos simplistas dizem que a inteligência artificial é uma tecnologia nova que tem sido utilizada de tal forma que os programas computacionais tornem os computadores mais espertos (GEVARTER, 1984). (SCHALKOFF, 1990) a define como "(...)gerar representações e procedimentos que automaticamente (autonomamente) resolvam problemas até agora resolvidos por humanos"

Algumas aplicações da inteligência artificial no campo da robótica estão relacionadas ao planejamento do caminho e movimento (navegação), visão computacional, controle, etc. (SCHALKOFF, 1990).

Conforme MCCOMB (1987) apud VIEIRA (1999), existem duas vertentes na definição de robôs móveis: uma que diz ser o robô completo, auto-contido, autônomo, que necessita instruções de seu mestre apenas ocasionalmente. A outra define robô como sendo qualquer equipamento que se mova por seus próprios meios, com o objetivo de executar tarefas próximas às humanas. As duas definições denotam que um robô móvel é capaz de manobrar livremente em seu ambiente, alcançando metas enquanto desvia de obstáculos, barreiras que lhe impeçam de se locomover pelo mundo.

Os robôs serão sempre conexões inteligentes entre percepção e ação (RICH & KNIGHT, 1994) e, para que estas conexões possam ser definidas e tenham um comportamento realmente inteligente, técnicas de inteligência artificial podem ser utilizadas.

Alguns autores classificam robôs inteligentes como aqueles que realizam decisões em tempo real, suportadas por algoritmos de inteligência artificial, baseados em sensores (KOREN, 1985), o que lhes possibilita trabalharem em ambientes completamente imprevisíveis de forma mais flexível. Na busca por comportamentos inteligentes em termos de processos computacionais, a implementação do simulador de agente robótico - descrita no capítulo cinco desta dissertação- aplica os conceitos de aprendizagem por reforço - técnica discutida no capítulo quatro - através do paradigma conexionista da Inteligência Artificial representada por redes neurais artificiais.

2.2 Arquiteturas robóticas e comportamento robótico

"Arquitetura Robótica é a disciplina devotada ao projeto de robôs altamente específicos e individuais a partir de uma coleção de blocos de construção comum". Versão modificada do conceito de arquitetura de computador, proposta por Stone, como pode ser visto em (ARKIN 1998).

ARKIN(1998) cita ainda uma outra referência, proposta por Hayes-Roth: "...um projeto abstrato de uma classe de agentes: um conjunto de componentes estruturais nos quais percepção, raciocínio e ação ocorrem; a funcionalidade e interface específica de cada componente e a topologia de interconexão entre os componentes"

Arquiteturas robóticas podem variar entre si profundamente e em vários aspectos, como a granularidade da decomposição dos comportamentos, a base para especificação dos comportamentos, programação e coordenação dos métodos empregados, dentre outros. Contudo, existem algumas características em comum a serem levadas em conta (ARKIN 1998):

- Ênfase na importância de acoplar sensação à ação;
- Decomposição em unidades contextualmente significativas (pares de comportamento ou situações)
- Não-utilização de conhecimento simbólico representacional.

Em relação a este último item, sua controvérsia é apontada como sendo útil e importante. No capítulo cinco, ARKIN(1998) versa sobre alguns requisitos representacionais para sistemas comportamentais, e justifica a utilização de conhecimento simbólico em razão de se obterem robôs que possam melhor agir, de posse de vários tipos úteis de conhecimento, tais como conhecimento espacial do mundo (espaço navegacional

do mundo), conhecimento de objeto (categorias ou tipos de objetos dentro do mundo), conhecimento comportamental (como reagir em diferentes situações), conhecimento de ego (limites das próprias ações dentro do mundo), além de outros.

2.3 Arquitetura *Subsumption*

Proposto por Rodney Brooks, este método baseado em comportamento puramente reativo consiste no uso de um sistema de controle em camadas (BROOKS 1990b). Nesta abordagem, os comportamentos são representados em camadas separadas e por sua vez podem agir em diferentes objetivos concorrentemente e assincronamente. Cada comportamento é representado, em mais baixo nível, por uma máquina de estados finitos aumentada(autômato). Cada autômato realiza uma ação e é responsável por sua própria percepção do mundo. Ainda, sinais de estímulos e/ou resposta podem ser adicionados ou inibidos por outros comportamentos ativos (ARKIN 1998).

O esquema de hierarquia de níveis (*levels*) de comportamento em camadas e sua relação com os sensores e atuadores do robô podem ser visto na figura 01 abaixo:

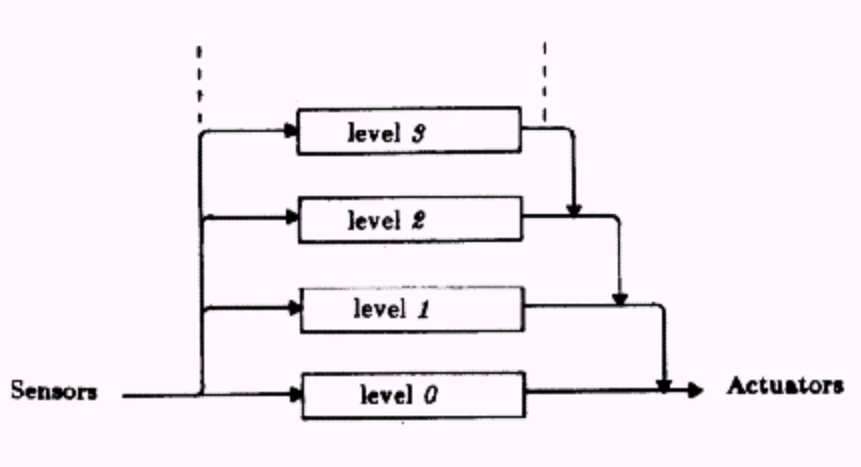

Figura 01: Representação da idéia básica da arquitetura Subsumption.
Fonte: http:// www.cis.tugraz.at/igi/STIB/WS98/ Bergmann/einleitung.htm

A idéia de comportamento sustenta os sistemas de controle reativo. Em contraste com sistemas planejados, onde a arquitetura de controle é dividida em tarefas funcionais (sentir, planejar ou processar, agir), sistemas reativos são compostos de múltiplas tarefas independentes que operam em paralelo. Cada comportamento é responsável por uma tarefa em particular, como evitar obstáculos, seguir parede, por exemplo. Cada comportamento processa sua própria informação sensorial, suas próprias entradas e lança seus próprios controles para os atuadores. Nesses termos, cada comportamento pode ainda desabilitar outros comportamentos, numa espécie de hierarquia. Esta propriedade de um comportamento poder cancelar um outro que possa gerar conflito com ele é conhecida como *subsumption*.

A arquitetura *Subsumption* e seu trabalho com robótica reativa foi uma das bases dos sistemas de robótica baseada em comportamento (MATARIC 1998), (BROOKS 1986).

2.4 Robótica baseada em comportamento

Basear as ações dos robôs em ações típicas de seres vivos é uma excelente técnica no sentido de implementar comportamentos inteligentes (MCFARLAND & BÖSSER, 1993).

A abordagem baseada em comportamento é uma metodologia para o projeto de arquiteturas e controladores inteligentes para robôs. Essa metodologia é baseada em uma filosofia inspirada biologicamente que favorece arquiteturas paralelas e descentralizadas.(MATARIC 1997)

Abordagens baseadas em comportamentos são intermediárias entre os extremos abordagens puramente reativas e abordagens baseadas em planejamento. (BROOKS&MAES 1990).

Embora apresente algumas propriedades de sistemas reativos, a abordagem baseada em comportamento não se limita a execução de mapeamentos funcionais simples. Comportamentos podem armazenar várias formas de estado e implementar vários tipos de representação.

Cada comportamento mantém um objetivo especifico. Por exemplo, o comportamento "evitar-obstáculo" mantém o objetivo de prevenir colisão com objetos do ambiente, ao passo que "buscar-alimento" persiste no objetivo de localizar um objeto específico no mundo.

Comportamentos são ativados em resposta às entradas de sensores e estados internos. O paralelismo pode ser explorado nesses sistemas, já que os comportamentos funcionam como subsistemas, tanto em velocidade de computação, como em resultado dinâmico. Como cada comportamento é um módulo ativo, esta dinâmica de interação surge da interação entre os comportamentos e dos comportamentos com o mundo externo. Nesse ponto, torna-se inevitável não se pensar em como coordenar estes comportamentos.

A coordenação de comportamentos ou arbítrio, ou seja a decisão de que comportamento executar em cada ponto no tempo é uma das preocupações e desafios centrais trazidos pelo paralelismo inerente à abordagem baseada em comportamentos. Em função da simplicidade de resolução a esta preocupação, pode-se promover prioridade de comportamentos ou seleção de comportamentos através da computação de alguma função votação de nível de comportamento, por exemplo. (MATARIC 1998)

Outra característica importante dos sistemas baseados em comportamento é sua abordagem à modularidade. Segundo MATARIC(1997), a execução de um comportamento não é simplesmente serializada. A metodologia organizacional de sistemas baseados em comportamento concerne a coordenação de múltiplos comportamentos, fazendo com que a arbitragem entre os comportamentos seja encarada como desafio central de muitos sistemas.

A nova proposta de pesquisa que a abordagem baseada em comportamento traz é o desafio de explorar meios de criar comportamentos inteligentes, estáveis, repetíveis sem contar com controle *top-down*, centralizado (MATARIC, 1997). Este novo caminho aberto a explorar as potencialidades do paralelismo dos comportamentos de forma inteligente, através de novos paradigmas deixa margem, para tanto, ao estudo e uso de redes neurais.

2.5 Arquitetura PiramidNet

Esta arquitetura apresentada por ROISENBERG em propõe que a inspiração biológica pode ser a fonte de mecanismos e soluções que, uma vez entendidos e implementados no ambiente computacional, permitirão entender, projetar e construir Redes Neurais Artificiais em Estruturas Modulares e Hierárquicas, capazes de serem aplicadas em robôs, a fim se desenvolver agentes inteligentes com alto grau de autonomia e utilidade.

Redes neurais de diferentes tipos dispostas em hierarquia visam controlar comportamentos reflexivos e reativos do agente, sendo que as redes diretas são responsáveis pela implementação dos comportamentos reflexivos básicos e as redes recorrentes pelas tarefas de coordenação e seleção destes comportamentos, intencionando obter um controle mais refinado do comportamento do agente autônomo, segundo SILVA(2001).

A arquitetura PiramidNet promove o conceito de prioridade de comportamento, não só com o intuito de separar os comportamentos, mas também para melhor coordena-los, no sentido de evitar conflitos e gerar tomada de decisão sobre qual comportamento deverá emergir em determinado instante. Uma figura representativa desta arquitetura pode ser vista abaixo.

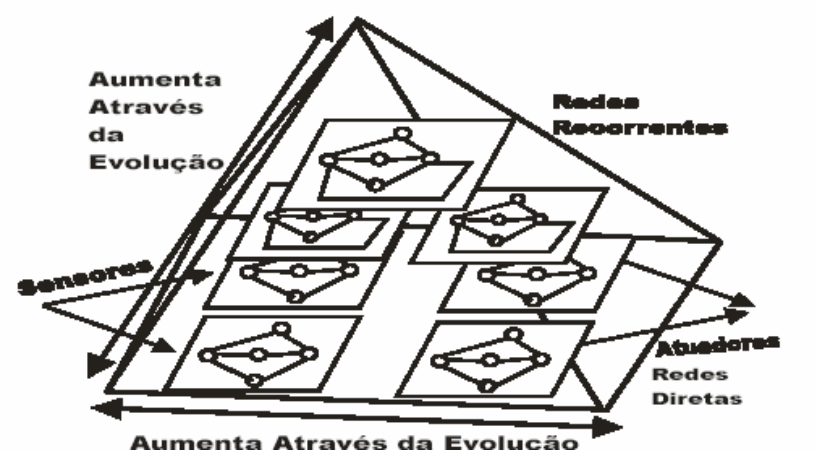

Figura 02: Representação da Arquitetura PiramidNet

Fonte: (SILVA 2001)

2.6 Aprendizagem em robôs

ARKIN (1998) afirma que a aprendizagem é parte essencial a um sistema inteligente. Para tanto, cita exemplos de ações que propiciam aumento de performance, através do incremento de aprendizagem em sistemas computacionais:

- Introduzir conhecimento novo (regras, comportamentos) dentro do sistema;
- Generalizar conceitos de múltiplos exemplos;
- Especializar conceitos para instâncias particulares;
- Reorganizar a informação dentro do sistema;
- Criar ou descobrir novos conceitos;
- Reusar experiências passadas.

A tarefa de aprender é particularmente difícil em robôs, devido às incertezas relacionadas às informações incompletas e ao ruído que os sensores podem captar nas condições de um ambiente que pode mudar dinamicamente. Existem ainda limitações tecnológicas concernentes a construção dos sensores e atuadores em relação a robôs que navegam em um mundo físico, por exemplo. (MATARIC 1990)

A aprendizagem por reforço é uma das técnicas mais populares nesse caso, porque permite ao robô melhorar seu comportamento através de tentativa-e-erro, recebendo um retorno de recompensa ou punição do ambiente, em resposta às ações realizadas. Esse método é melhor descrito no capítulo 04, já que constitui-se também tema de estudo desta dissertação.

3 Redes Neurais Artificiais

Uma rede neural artificial pode ser definida como "uma coleção de processadores paralelos conectados em forma de um grafo dirigido e organizado de modo que esta estrutura se adapte ao problema considerado". (FREEMAN&SKAPURA, 1991)

Segundo BITTENCOURT(1998), algumas das características que tornam a metodologia de redes neurais uma abordagem interessante com vistas à solução de problemas são as citadas a seguir:

1. Capacidade de aprender através de exemplos e de generalizar esta aprendizagem de maneira a reconhecer instâncias similares que nunca haviam sido apresentadas como exemplo;

2. Elevada imunidade ao ruído, isto é, o desempenho de uma rede neural não entra em colapso em presença de informações falsas ou ausentes, embora sofra uma piora gradativa;

3. Não requer conhecimento a respeito de eventuais modelos matemáticos dos domínios de aplicação;

4. Apresentam bom desempenho em tarefas mal-definidas, onde falta o conhecimento explícito sobre como encontrar uma solução. Do ponto de vista de se trabalhar com agentes autônomos, explorando um ambiente desconhecido, esta característica torna-se de extrema importância e justifica uma das razões para o uso de redes neurais nesse problema específico.

3.1 Inspiração biológica

A origem da teoria das redes neurais remonta aos modelos matemáticos de neurônios biológicos. Santiago Ramón y Cajal postulou sobre a comunicação das células nervosas pela sinapse e que a interconexão entre os neurônios não seria feita ao acaso e sim seria altamente específica e estruturada. A descrição de Cajal das estruturas cerebrais voltou as pesquisas para o desconhecido campo das estruturas formadas por grupos de neurônios(BARRETO,2001). Esses estudos, deram início, de certa forma, a visão das redes neurais como um conjunto de estruturas, o que se traduz na primeira noção básica pressuposta para a utilização da inspiração biológica na computação de redes de células paralelas.

Muito simplificadamente, um neurônio biológico é formado por um corpo celular (soma), o qual contém o núcleo da célula, por diversos dendritos, pelos quais os impulsos elétricos são recebidos e por um axônio, onde os impulsos elétricos são enviados. Os neurônios são interligados através de regiões eletroquimicamente ativas denominadas sinapses. São esses pontos de contato que permitem a propagação dos impulsos nervosos de uma célula a outra. As sinapses podem ser excitatórias ou inibitórias. "As sinapses excitatórias cujos neuro-excitadores são os íons sódio permitem a passagem da informação entre os neurônios e as sinapses inibitórias, cujos neuro-bloqueadores são íons potássio, bloqueiam a atividade da célula, impedindo ou dificultando a passagem da informação".(ROISENBERG 2001[N. do A.])

Embora as redes neurais artificiais tenham inspiração em neurônios biológicos, é importante lembrar que suas semelhanças são mínimas. A intenção, originalmente era imitar a realidade biológica. Mesmo que muitos pesquisadores atuais ainda discordem disso, a pesquisa atual é motivada por dois fatores, citados a seguir por BARRETO(2001):

[N. do A.] ROISENBERG, Mauro. Apostila compilada da disciplina de Redes Neurais I. Disciplina oferecida no curso de Pós-Graduação em Ciência da Computação da Universidade Federal de Santa Catarina

- Modelar o sistema nervoso com precisão suficiente de modo a poder-se observar um comportamento emergente, que sendo este semelhante ao comportamento de um ser vivo modelado, possa servir de apoio às hipóteses usadas na modelagem;

- Construir computadores com alto grau de paralelismo.

3.2 Características gerais de uma rede neural

Uma rede neural artificial é composta por várias unidades de processamento. Essas unidades, os neurônios, estão associados a determinado peso. As unidades fazem operações apenas sobre seus dados locais, que são entradas recebidas pelas suas conexões. As interações entre os neurônios gera o comportamento inteligente da rede.

O neurônio artificial é uma estrutura lógico-matemática que procura simular a forma, o comportamento e as funções de um neurônio biológico. Assim sendo, os dendritos foram substituídos por *entradas*, cujas ligações com o corpo celular artificial são realizadas através de elementos chamados de *peso* (simulando as sinapses). Os estímulos captados pelas entradas são processados pela *função de soma*, e o limiar de disparo do neurônio biológico foi substituído pela *função de transferência*. (TAFNER, 1999). O esquema representativo de um neurônio artificial pode ser visto a seguir:

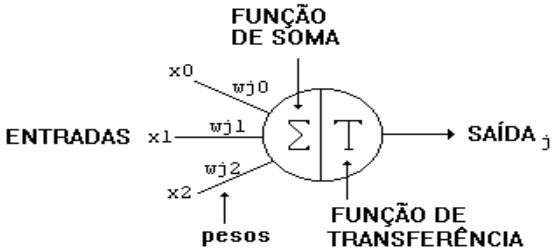

Figura 03: Esquema representativo de um neurônio artificial

A operação de uma unidade de processamento, proposta por McCullock e Pitts em 1943, pode ser resumida da seguinte maneira: cada sinal de entrada é multiplicado por um peso, que indica a sua influência na saída da unidade. Se a soma ponderada dos sinais produzidos exceder a um certo limite (*threshold*), a unidade produz uma determinada resposta de saída.

Uma arquitetura neural[2] é tipicamente organizada em camadas, estas interligadas. Por definição, uma camada é um conjunto de neurônios recebendo as entradas em um mesmo local e tendo as saídas em um mesmo local.(BARRETO,2001).

As redes neurais se diferenciam, também, pela topologia e tipo de treinamento.

Quanto à topologia, as redes neurais podem ser classificadas em dois grandes grupos:

*Redes Recorrentes: São as redes cíclicas, com realimentação. Ou seja, a saída do processamento atual é novamente aplicada no processamento seguinte, funcionando como uma nova entrada. Um exemplo de rede recorrente está explicitado na figura 04, abaixo.

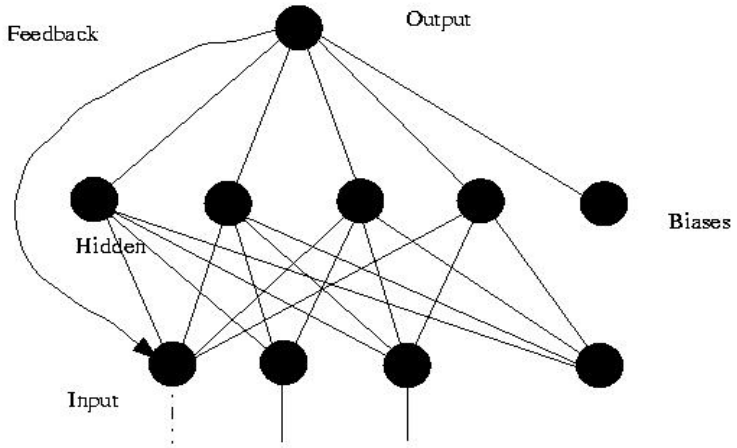

Figura 04: Exemplificação de uma arquitetura de rede neural recorrente
Fonte: http://openai.sourceforge.net/docs/nn_algorithms/networksarticle/

[2] Para um entendimento mais aprofundado de redes neurais, sugere-se (BARRETO,2001),(BITTENCOURT,1998),(LOESCH&SARI,1996),(ZSOLT,1996)

* Redes Diretas: Vistas como grafos, são aquelas cujos grafos não tem ciclos. O processamento se dá em um sentido somente, sem retroalimentação. São de mais fácil implementação e portanto mais utilizadas. Um exemplo de rede direta pode ser visto na figura 05.

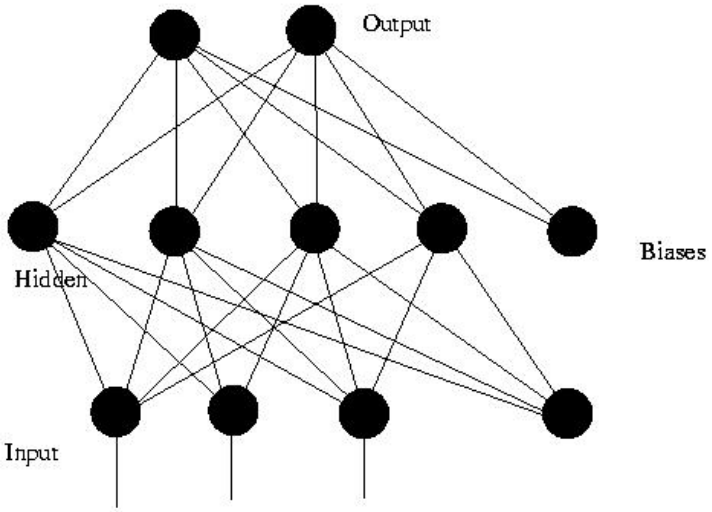

Figura 05: Exemplificação de uma arquitetura de rede neural direta
Fonte: http://openai.sourceforge.net/docs/nn_algorithms/networksarticle/

A propriedade mais importante das redes neurais é a habilidade de aprender de seu ambiente e assim obter uma melhora de desempenho. Isso é feito através de um processo interativo de ajustes aplicado a seus pesos, o que é chamado treinamento. Durante o treinamento, os pesos de suas conexões são ajustados de acordo com os padrões apresentados. Podemos assim dizer que ocorre aprendizagem através de exemplos. A aprendizagem ocorre quando a rede neural atinge uma solução generalizada para uma classe de problemas.

Denomina-se algoritmo de aprendizagem a um conjunto de regras definidas para a solução de um problema de aprendizagem. Existem muitos tipos de algoritmos de

aprendizagem específicos para determinados modelos de redes neurais, estes algoritmos diferem entre si principalmente pelo modo como os pesos são modificados.

Outro fator importante é a maneira pela qual uma rede neural se relaciona com o ambiente externo. Nesse contexto, os paradigmas de aprendizagem podem ser separados em dois grandes grupos:

- Aprendizagem Supervisionada, quando é utilizado um agente externo que indica à rede a resposta desejada para o padrão de entrada;

- Aprendizagem Não Supervisionada, quando não existe um agente externo indicando a resposta desejada para os padrões de entrada. No estudo apresentado neste trabalho, tratamos aprendizagem por reforço como pertencendo a esta classe, ainda que alguns autores classifiquem este tipo de aprendizagem em especial como uma classe em si ou ainda como pertencente a uma nova classe intermediária, denominada aprendizagem semi-supervisionada.

3.3 O Dilema da Plasticidade-Elasticidade e as redes ART

"Como pode um sistema de aprendizagem permanecer plástico, ou adaptativo, em resposta a eventos significantes, e ainda permanecer estável em respostas a eventos irrelevantes? Como alcançar estabilidade sem rigidez e plasticidade sem caos? Como preservar conhecimentos aprendidos anteriormente, enquanto continuam-se aprendendo novas coisas?"

O parágrafo anterior, extraído de LOESCH&SARI(1996) diz respeito ao dilema da plasticidade-elasticidade. Inúmeras topologias de redes neurais têm falhado ao tentar

transpor esse dilema, pois através delas, aprender um novo padrão significa apagar o anterior. Nesse ponto, para dar suporte a esse problema, Grossberg e Carpenter lançaram-se à pesquisa de um novo modelo, conhecido como ART (Adaptive Resonance Theory) (FREEMAN&SKAPURA,1991). A rede neural ART é tida como um classificador de vetores entrada binária e sua topologia de rede possui um mecanismo de retorno entre a camada competitiva e a camada de entrada da rede.

Na física, a ressonância ocorre quando uma vibração de pequena amplitude da própria freqüência causa uma vibração de grande amplitude em um sistema elétrico ou mecânico. Em uma rede ART, a informação na forma de saídas de elemento de processamento reverbera de volta e pra frente entre as camadas. Se os próprios padrões são desenvolvidos, uma oscilação estável se sucede, o que é a equivalência da ressonância em rede neural. (FREEMAN&SKAPURA,1991)

É durante o período ressonante que a aprendizagem ou a adaptação podem ocorrer. Um estado ressonante é obtido de duas formas. Se a rede aprendeu a reconhecer previamente um vetor de entrada, então o estado ressonante será rapidamente recuperado quando o vetor de entrada é apresentado. Durante a ressonância, o processo de adaptação reforçará a memória do padrão armazenado. Se o vetor de entrada não é imediatamente reconhecido, a rede pesquisará rapidamente entre os padrões armazenados procurando por um padrão equivalente. Se este não for encontrado, a rede entrará em estado ressonante e como conseqüência, o novo padrão será armazenado pela primeira vez.

A rede ART mantém equilíbrio entre as propriedades de plasticidade (discriminação) e de estabilidade (generalização) (LOESCH&SARI,1996). Ou seja, é capaz de criar uma nova categoria de padrões, quando estimulada por padrões não reconhecidos e ainda agrupar padrões similares na mesma categoria, quando reconhecidos. Uma regra de similaridade que define onde agrupar os padrões é determinada por um grau de semelhança entre um padrão dado e padrões previamente armazenados.

Assim, a rede responde rapidamente a dados aprendidos previamente e ainda é capaz de aprender quando novos dados são apresentados.

Modelos ART são mais adaptáveis para aprender e responder em tempo real, para um mundo não-estacionário, com um número ilimitado de entradas, ou até que seja utilizada toda sua capacidade de memória, como confirma LOESCH&SARI (1996).

A arquitetura da rede ART, conforme pode ser vista na figura 06 apresenta duas camadas de neurônios chamadas de F1 (comparação/entrada) e F2(reconhecimento e estocagem). Os nós de uma camada são totalmente interconectados aos nós da outra camada. As funções de controle, ganho G1 e G2 e Reset são necessárias ao treinamento e classificação dos padrões.

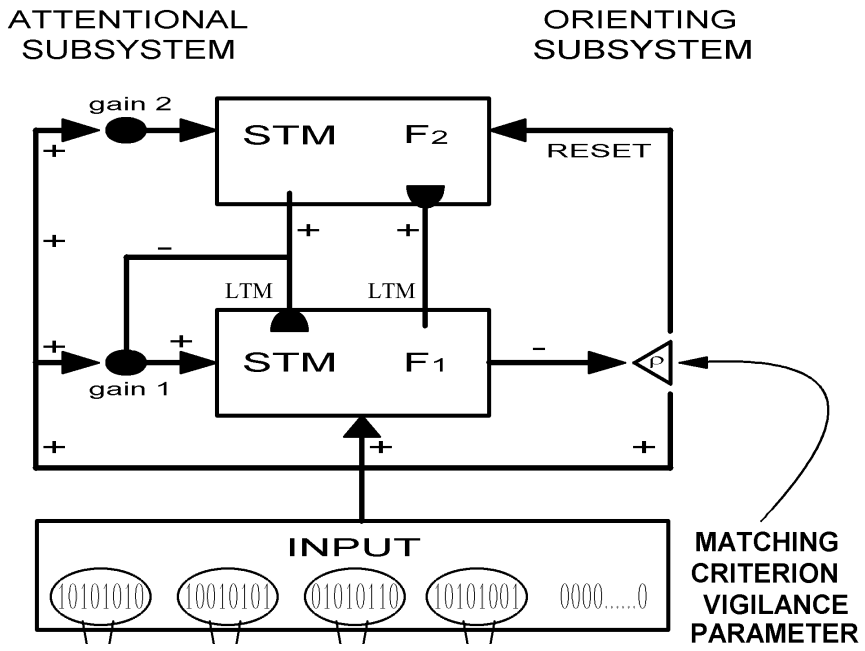

Figura 06: Exemplificação de uma arquitetura de rede neural ART

Fonte: http://ailab.kyungpook.ac.kr/icips/hkang.html

O funcionamento de uma rede neural ART pode ser entendido, dividindo-o em um processo de classificação hipotético em etapas, como proposto por LOESCH&SARI (1996). A saber, as quatro etapas constituem as fases de reconhecimento, comparação, pesquisa e treinamento, embora este último faça parte da própria dinâmica da rede, que não distingue entre fase de operação normal e aprendizagem. Isto é, faz parte do todo, não podendo ser considerada uma etapa em separado.

Antes de que se explicitem as etapas, apresenta-se a nomenclatura para os símbolos utilizados:

x= vetor de entrada aplicado a rede, que contêm o padrão a ser classificado;

p= vetor de pesos *top-down* que entra na camada de comparação, vindo do neurônio vencedor da camada de reconhecimento;

c= vetor de saída da camada de comparação;

r= vetor de saída da camada de reconhecimento;

Primeira etapa: Reconhecimento

O vetor de entrada **x** com os padrões a serem analisados (classificados) é aplicado. Um ou mais componentes desse vetor precisam ser 1, o que implica que os controles de ganho G1 e G2 também serão 1. Para se ter saída 1, pelo menos duas das três entradas dos neurônios da camada de comparação (F1) precisam ser 1. Está-se tratando da regra implementada por Grossberg e Carpenter conhecida como regra dos 2/3. Cada neurônio da camada de reconhecimento (F2) tem um vetor peso associado, que constitui em padrão armazenado. Somente o neurônio com um peso que melhor combine com o vetor de entrada, enquanto os demais são inibidos. Esse neurônio vencedor terá maior ativação e é por tanto conhecido como neurônio ressonante. O processamento feito nessa camada é algo um pouco mais complexo, mas semelhante ao feito na camada de entrada de uma rede de arquitetura counterpropagation, segundo FREEMAN&SKAPURA (1996).

Segunda Etapa: Comparação

O neurônio disparado na camada de reconhecimento vai para a camada de comparação (F1). O seu sinal de saída **r** é 1. O controle de Ganho G1 é inibido. Assim, pela regra dos 2/3, os únicos neurônios a disparar são os que recebem valores de entrada 1 do vetor de entrada **x** e do vetor **p**. Caso exista uma diferença substancial entre **x** e **p**, ou seja, poucas coincidências, poucos neurônios da camada de comparação irão disparar e **c** conterá muitos zeros, enquanto **x** menos, o que indica que o padrão **x** sendo retornado não é aquele procurado e o neurônio disparado na camada de reconhecimento será inibido. Esta inibição é feita pelo reset, que compara o vetor de entrada **x** com o vetor **c** e causa o sinal de reset caso seu grau de similaridade seja menor que o nível de vigilância. A medida de similaridade é computada dividindo-se o número de unidades em **c** pelo número de unidades em **x**, podendo variar de 0 a 1.

Terceira etapa: Pesquisa

Caso não ocorra o sinal de reset, a combinação é tida como adequada e a classificação é terminada. Caso contrário, outros padrões estocados precisam ser procurados para buscar uma combinação melhor. No último caso, a inibição do neurônio disparado causa que todos os componentes do vetor **r** retornem a 0, G1 fica em 1 e o padrão de entrada **x** aparece outra vez em **c**. Como resultado, um neurônio diferente vence na camada de reconhecimento e um padrão estocado p diferente é retornado para a camada de comparação. Se p não combinar com **x**, esse neurônio da camada de reconhecimento disparado é também inibido. Este processo repete-se, neurônio a neurônio, até que um dos eventos abaixo ocorra:

- um padrão armazenado é descoberto que combine com x e possua nível de similaridade maior que o parâmetro de vigilância;
- se todos os padrões estocados foram experimentados, existe desigualdade do padrão de entrada e todos os neurônios da camada de reconhecimento são inibidos;
- não existe um padrão estocado que combine com o vetor de entrada. Esgota-se a capacidade da rede de forma que o novo padrão não é alocado.

Treinamento

O treinamento em uma rede ART é não supervisionado e constitui-se o processo no qual um conjunto de vetores de entrada é apresentado seqüencialmente pela entrada da rede, e os pesos da rede são ajustados para que vetores similares ativem o mesmo neurônio da camada de reconhecimento.

Como posto em LOESCH&SARI(1996), "a rede é treinada para ajustar os pesos top-down e bottom-up para que a aplicação de um padrão de entrada leve a rede a ativar o elemento de processamento – ou neurônio- da camada de reconhecimento associado com um padrão similar. Além disto, o treinamento é realizado de uma maneira que não se destrua os padrões aprendidos anteriormente, característica controlada pelo parâmetro de vigilância. Um novo padrão de entrada (não visto ainda pela rede) irá falhar para combinar padrões estocados dentro de uma tolerância imposta pelo nível de vigilância, causando um novo padrão estocado a ser formado. Um padrão de entrada suficientemente parecido com um padrão estocado não irá formar outro novo, ele irá simplesmente modificar um com o qual ele se pareça. Com um ajuste adequado do nível de vigilância, novos padrões já aprendidos e instabilidades temporais são evitados."

O algoritmo abaixo, extraído de (VALLE FILHO et al, 1999) ilustra o treinamento de uma ART:

1) Repita EP vezes (épocas)
  2) Repita para cada vetor de entrada
    3) Ativar a camada F1;
    4) Atualizar F1;
    5) Calcular sinais para enviar a F2;
    6) Enquanto RESET=V faça
      7) Encontrar o vencedor;

8) Calcular RESET;

9) se RESET=V {se o vencedor for rejeitado}

então inibir J;

Fim_enquanto passo 6.

10) Atualizar F1;

11) Repita APR {aprendizagem}

12) Atualizar pesos;

13) Atualizar F1;

Fim_repita passo 11.

Fim_repita passo 2.

Fim_repita passo 1.

3.3.1 Tipos de redes ART

Os tipos de redes ART mais usados são ART1 e ART2. Os dois funcionam de maneira análoga, como explicitado acima, só que a primeira se caracteriza por trabalhar com entradas binárias. Já a ART2 admite padrões de entrada analógicos.

As redes ART podem ser também do tipo supervisionada, quando utilizam um agente externo que indica a resposta desejada para o padrão de entrada, ou não-supervisionada (auto-organizável), quando não utiliza um agente externo para indicar a resposta desejada (VALLE FILHO et al 1999).

O modelo Fuzzy ART também incorpora o modelo ART1 e como o nome evidencia, trabalha com cálculos de lógica Fuzzy. Esses cálculos, oriundos da teoria de conjuntos Fuzzy são reduzidos a cálculos binários quando as variáveis Fuzzy tornam-se valores binários. Se comparada a ART1, o modelo Fuzzy ART executa uma operação adicional de pré-processamento dos padrões de entrada

Segundo VALLE FILHO et al (1999), as redes ARTMAP combinam uma ou mais redes ART1. Nelas, o parâmetro de vigilância, antes estático, trocado agora por uma variável dinâmica que aumenta e diminui em resposta ao grau de dificuldade apresentado por uma tarefa de reconhecimento de padrões. Como são várias redes ART1, também trabalham com vetores binários.

Acrescentando supervisão a esse modelo, deriva-se o modelo Fuzzy ARTMAP. Este tipo de rede apresenta um mapa associativo que funciona como memória que controla seus dois módulos Fuzzy ART. O uso deste mapa enquadra a rede ART na categoria das redes preditivas (GUAZELLI, 1994). A arquitetura ARTMAP é capaz de promover aprendizagem supervisionada em tempo real, como pode ser visto detalhadamente em CARPENTER et al (1991).

Existem ainda muitas outras adaptações de redes ART que incorporam características distintas. Abaixo, citam-se algumas destas adaptações extraídas de (ART CLEARINGHOUSE 2002), de onde é possível se chegar a outras referências sobre cada tipo. A menção a estes tipos especializados de rede ART abaixo é meramente ilustrativa, com o propósito de mostrar a diversidade de redes ART especializadas existentes.

* Gaussian ARTMAP, os quais implementam aprendizagem rápida de mapeamentos multidimensionais com ruído;

* LAPART: arquitetura de rede neural ART para verificação de seqüência de padrões através de inferenciação;

* MART: rede neural ART Multicanal, para classificação adaptativa de padrões através de múltiplos canais de entrada;

* PROBART: rede fuzzy ARTMAP modificada por informação probabilística, que permite aprendizagem por mapeamento de ruído;

* HART: redes modulares ART que aprendem clusters(grupos) hierárquicos de seqüências arbitrárias de padrões de entrada.

4 Aprendizagem por Reforço

4.1 Caracterização

Para MITCHEL (1997), aprendizagem por reforço é o estudo de como um agente autônomo, que percebe e atua no seu ambiente, pode aprender a escolher ações adequadas a fim de atingir os seus objetivos. KAELBLING (1996) cita que aprendizagem por reforço preocupa-se com agentes que aprendem um determinado comportamento através de interações de tentativa-e-erro com um ambiente dinâmico.

Em termos gerais, aprendizagem por reforço baseia-se na idéia de que a tendência por parte de um agente a produzir uma determinada ação dever ser reforçada se ela produz resultados favoráveis e enfraquecida se ela produz resultados desfavoráveis.

Em um sistema de aprendizagem por reforço, o estado do ambiente é representado por um conjunto de variáveis (espaço de estados) percebidas pelos sensores do agente. Uma ação escolhida pelo agente muda o estado do ambiente e o valor desta transição de estados é passado ao mesmo através de um sinal escalar de reforço, denominado recompensa. O objetivo do método é levar o agente a escolher a seqüência de ações que tendem a aumentar a soma dos valores de recompensa. O agente se move autonomamente no espaço de estados interagindo com o ambiente e aprendendo sobre ele através de ações por experimentação. Cada vez que o agente realiza uma ação, uma entidade externa treinadora (alguns autores se referem a ela como um professor (SUTTON, 1998) ou ainda crítico (KAELBLING, 1996)) ou mesmo o ambiente pode lhe dar uma recompensa ou uma penalidade, indicando o quão desejável é alcançar o estado resultante. Desta forma, o reforço nem sempre significa avanço, como pode também inibir o agente em relação à ação executada. A figura 07 abaixo representa um esquema genérico da idéia de aprendizagem por reforço.

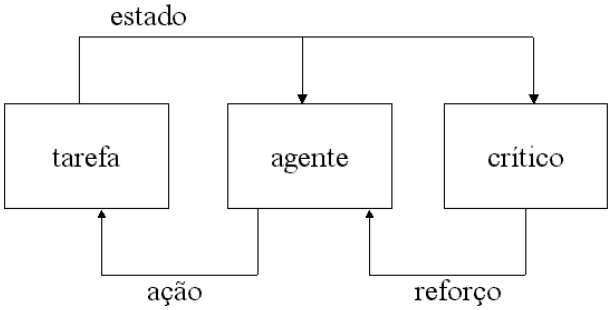

Figura 07 : Representação genérica da idéia de aprendizagem por reforço.

Aprender por reforço é aprender o que fazer: ter conhecimento da(s) meta(s) a se atingir, mapear situações em ações, de forma a maximizar um sinal de recompensa ou punição numérico. Só não como fazer. Ao agente aprendiz não é dito quais ações tomar, mas ao invés disso descobrir quais ações perfazem um maior número de recompensas ou ainda um menor número de punições.

A aprendizagem por reforço é diferente de aprendizagem supervisionada, o tipo de aprendizagem estudado na maioria das pesquisas atuais em aprendizagem de máquina e reconhecimento de padrão estatístico. Aprendizagem supervisionada é a aprendizagem provida por um supervisor externo de conhecimento. Este é um tipo importante de aprendizagem, mas não é muito adequado para aprender por interação, segundo SUTTON&BARTO (1998). Em problemas interativos é freqüentemente impraticável obter exemplos de comportamento desejado que seja tanto correto como representativo para todas as situações nas quais o agente precisa agir. Em território desconhecido um agente deve poder aprender a partir de sua própria experiência.

A aprendizagem por reforço é uma técnica que pode ser feita em tempo real (on-line), ou seja, em constante adaptação, não necessitando de um treinamento prévio ou off-line do sistema. Essa vantagem torna-se importante no intuito de se aplicar esse método a sistemas

robóticos que navegam em um ambiente incerto. Conforme KAELBLING (1996), em aprendizagem por reforço não existe apresentação de conjuntos de exemplos. São informados ao agente – depois deste escolher uma ação – a recompensa imediata e o estado subseqüente, mas não qual ação teria sido melhor de acordo com seus interesses.

Outra característica chave da aprendizagem por reforço é que ela considera explicitamente todo o problema de um agente direcionado a meta que interage com um ambiente incerto. Isto está em contraste com muitas abordagens que se dirigem a sub-problemas sem se dirigir como eles poderiam se ajustar em um quadro maior.

Segundo SUTTON&BARTO (1998), existem três classes fundamentais de métodos para a solução de problemas de aprendizagem par reforço: programação dinâmica, métodos de Monte Carlo e aprendizagem por diferença temporal - *Temporal Difference* (TD). Os métodos de programação dinâmica são matematicamente bem desenvolvidos, no entanto necessitam de um modelo completo e preciso do ambiente. Os métodos de Monte Carlo não necessitam de um modelo, possuem conceitos mais simples, mas não são adequados para uma computação incremental passo a passo. Os métodos baseados em TD, par sua vez, também não necessitam de um modelo e são métodos incrementais, porém são mais complexos de se analisar.

4.2 Problemas que a aprendizagem por reforço envolve

Um dos desafios que surgem em aprendizagem por reforço e não em outros tipos de aprendizagem é o dilema clássico entre utilização (ações já aprendidas) e investigação (ações a aprender) - *exploitation versus exploration*. Ou ainda conhecido como dilema Exploração X Descoberta. Para obter uma quantidade maior de recompensa, um agente deve preferir ações que tentou no passado efetivas na produção de recompensa. Mas para

descobrir tais ações deve tentar ações não realizadas anteriormente. O agente tem que utilizar o que já conhece para obter recompensa, mas também tem que explorar(investigar) para fazer melhores seleções de ação no futuro. O dilema em relação à tarefa é que nem a utilização nem a investigação podem ser continuadas exclusivamente sem falhar. O agente tem que tentar uma variedade de ações e progressivamente deve favorecer as que parecem ser melhores. Em uma tarefa estocástica, cada ação deve ser tentada muitas vezes para calcular sua recompensa média esperada de forma confiável.

Outro grande problema de sistemas de aprendizagem por reforço é a tarefa de crédito temporal, ou seja, como punir ou recompensar uma ação quando poderia ter alcançando mais efeitos? Fazendo uma analogia com um jogo de xadrez, ao longo de uma partida, como saber quais movimentos específicos são responsáveis pela vitória ou derrota?

4.3 Elementos da Aprendizagem por Reforço

Além do agente e do ambiente, pode-se identificar elementos principais para um sistema de aprendizagem por reforço: uma política, uma função de recompensa, uma função de valor e, opcionalmente, um modelo do ambiente. SUTTON&BARTO (1998)

Uma política define a maneira do agente da aprendizagem de se comportar em um determinado momento, qual ação deve ser realizada em cada estado. Uma política é um mapeamento de estados percebidos do ambiente em ações a serem tomadas quando o agente estiver nesses estados. Corresponde ao que em psicologia seria chamado de um conjunto de regras de estímulo-resposta ou associações. Quando este mapeamento maximiza a soma das recompensas, tem-se o que se chama política ótima. Em alguns casos a política pode ser uma simples função, considerando que em outros pode envolver computação extensa como um processo de busca.

Uma função de recompensa ou de reforço define a meta em um problema de aprendizagem por reforço. Ela mapeia estados percebidos (ou pares ação-estado) do ambiente para um único número, uma recompensa, indicando o desejo intrínseco do estado. O objetivo exclusivo de um agente de aprendizagem por reforço é maximizar a recompensa total que recebe (no fim do percurso, no final das contas). A função de recompensa define o que são eventos bons e maus para o agente.

A função de recompensa necessariamente deve ser fixada. Porém, pode ser usado como uma base para mudar a política. Por exemplo, se uma ação selecionada pela política é seguida por uma baixa recompensa então a política pode ser mudada para selecionar alguma outra ação nesta situação no futuro. Em geral, funções de recompensa podem também ser estocásticas. (SUTTON&BARTO, 1998)

Considerando que uma função de recompensa indica o que é bom em uma sensação imediata, uma função de valor especifica o que é bom no final das contas. O valor de um estado é a quantidade total de recompensa que um agente pode esperar acumular em cima do futuro a partir daquele estado.

Considerando que recompensas determinam o desejo imediato, intrínseco de estados do ambiente, valores indicam o desejo a longo prazo de estados depois de levar em conta os estados que são prováveis seguir, e as recompensas disponíveis nesses estados. Por exemplo, um estado poderia sempre produzir uma baixa recompensa imediata, mas ainda ter um valor alto porque é seguido regularmente por outros estados que produzem recompensas altas.

Recompensas são de certo modo primárias, considerando que valores, como predições de recompensas, são secundários. Sem recompensas não poderia haver nenhum valor, e o único propósito de calcular valores é alcançar mais recompensa. Não obstante, são valores com os quais estamos preocupados quando fazendo e avaliando decisões. Escolhas de ação

são feitas com base nos julgamentos de valor. Buscamos ações que provocam estados de valor mais alto, não recompensa mais alta, porque estas ações obtêm a maior quantia de recompensa ao longo de todo percurso (SUTTON&BARTO, 1998) , (KAELBLING, 1996).

É muito mais difícil determinar valores do que determinar recompensas. (SUTTON&BARTO, 1998). Recompensas são basicamente determinadas diretamente pelo ambiente, mas valores devem ser calculados e devem ser re-calculados das sucessões de observações que um agente perfaz ao longo de sua interação. De fato, o componente mais importante de quase todos os algoritmos de aprendizagem por reforço é um método para estimar valores eficientemente.

4.4 Técnicas de aproximação da função valor

No intuito de se aproximar a função de valor, a idéia fundamental é que, como o sistema deve aprender com a experiência, se uma ação executada causou uma situação indesejada, o sistema aprenda a não mais executar tal ação naquela situação. Para isso ser possível, utilizam-se conceitos de programação dinâmica, onde o processo de aprendizagem consiste em encontrar uma solução que, por sucessivas atualizações dos valores dos estados de uma tabela, estes tenham convergido, ou seja, não mudem mais a partir de dado momento. Em se tratando de aproximação de função em aprendizagem por reforço destaca-se o método de aprendizagem por Diferença Temporal – *Temporal Difference* (TD) e uma técnica baseada neste método chamada *Q-Learning*. Ambos são explicitados a seguir.

4.5 Métodos de Diferença Temporal

Métodos de diferença temporal ou Temporal Difference (TD) são algoritmos de aprendizagem gerais para fazer predições a longo prazo sobre sistemas dinâmicos. Eles estão baseados em funções de valor estimadas, funções de estados V(s) ou pares ação-estado Q(s,a) que estimam quão bom é para o sistema de aprendizagem estar em um determinado estado ou entrar uma certa em ação em um determinado estado. Tais ações de valor guiam o mecanismo de seleção de ação, a política para equilibrar utilização e investigação para maximizar recompensa com o passar do tempo, e permitir que o sistema de aprendizagem alcance sua meta. Em outras palavras, quanto mais próximo da convergência do método, mais o agente tem certeza de qual ação tomar em cada estado.

Métodos TD apresentam vantagens sobre outros métodos também utilizados em aprendizagem por reforço como Programação Dinâmica e método de Monte Carlo[3] . TD não necessita de uma modelagem baseada em processos de decisão markovianos[4] ao ambiente, nem das transições de probabilidade de seus estados. TD pode ser implementado de forma totalmente incremental para aplicações on-line, segundo (TUPAC&VALDIVIA).

4.5.1 Método Q-Learning

É um método que calcula uma política ótima para aprendizagem por reforço baseado em TD. Neste método, a cada iteração o agente observa o estado atual **i**, realiza uma ação **a**, observa o novo estado **i'**, e recebe uma recompensa imediata **r**. O agente tem interesse não apenas nas recompensas imediatas, como também na recompensa total descontada, onde as

---

[3] Para saber mais sobre esses métodos, consultar (TUPAC&VALDIVIA), (SUTTON&BARTO, 1998)

[4] Idem a nota 2

recompensas recebidas em **n** iterações mais tarde tem menos valor do que as recebidas no estado atual, por um fator de desconto γ, onde $0 \leq \gamma < 1$.

$$R = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots$$

O agente tenta determinar Valores-Qualidade (Q-Values) para cada par (i,a). A cada passo, deve-se aproximar a expressão

$$Q(i,a) \rightarrow (R(i,a) + \gamma \max Q(i',a') - Q(i,a)), \text{ onde}$$

- Q(i,a) é o valor (qualidade) da ação a no estado i;
- R(i,a), a recompensa da ação a no estado i;
- γ o fator de desconto

e:

- "$Q(i,a) \rightarrow x$" significa ajustar a estimativa Q(i,a) na direção de x.

Em (TUPAC&VALDIVIA), encontra-se um pseudo-código para explicitar o algoritmo Q-Learning :

```
Inicializar Q(s,a) arbitrario
  Repete (para cada episodio)
  Inicializa s
        Repete (para cada passo do episodio)
        Escolhe a para s desde Q
        Toma ação a, observa r e s'
        Aplica equação Q-Learning (ver acima)
        s←s'
Até s ser o estado terminal
```

5 Proposta e Análise de Arquiteturas

5.1 Descrição geral

Após um levantamento bibliográfico sobre agentes autônomos, redes neurais artificiais e aprendizagem por reforço, a próxima etapa deste trabalho tem como objetivo desenvolver uma série de arquiteturas neurais com intuito de investigar como adaptar as idéias de aprendizagem por reforço para um agente autônomo em tempo real controlada por redes neurais. A aprendizagem está concentrada no fato de o agente ser capaz de reconhecer uma situação já experimentada e utilizar essa experiência adquirida para resolver uma nova situação idêntica da melhor forma possível, neste caso, se movimentar por segmentos livres, ou que não possuam obstáculos que o impeçam de se locomover. Exemplos de situações podem ser vistos nas figuras 8a e 8b abaixo. Na figura 8a, o agente possui obstáculos nas direções oeste, sul e leste, enquanto que na figura 8b os obstáculos estão nas direções oeste e norte.

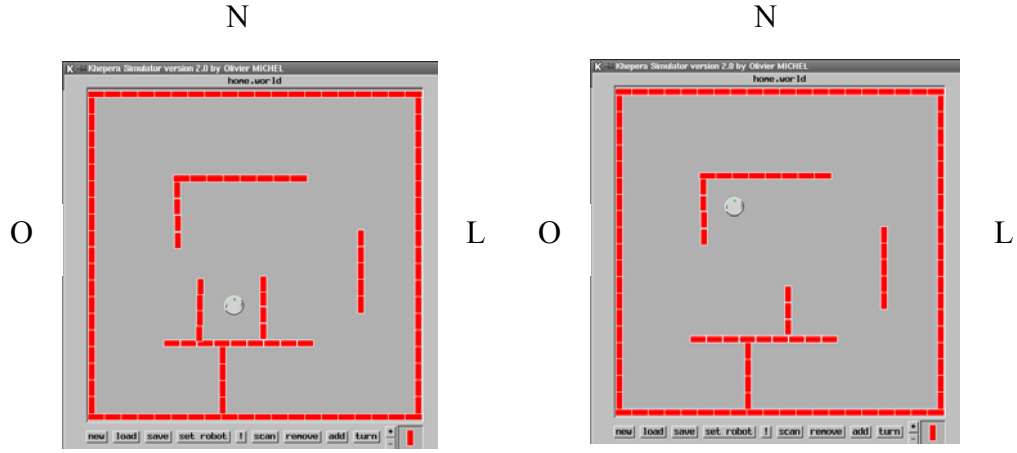

Figura 8a e 8b: Simulador de robô Khepera exemplificando possíveis situações

A técnica de aprendizagem por reforço, por ser uma técnica de aprendizagem por tentativa e erro pressupõe uma aprendizagem em tempo real, ou seja, o agente recebe um reforço

negativo quando não puder se mover no ambiente, ou seja, quando detectasse a presença de obstáculo(s) na(s) célula(s) adjacente(s) aos seus sensores.

Para isso, foi concebido um ambiente de teste para simulação de diversos tipos de mundos, com disposições de obstáculos de modos diferentes, apresentando diversos graus de complexidade com o intuito de se avaliar os modelos propostos bem como a capacidade cognitiva do agente.

O agente que atua no simulador apresenta quatro sensores e quatro atuadores, como pode ser visto na figura 09. Os sensores são responsáveis pela aquisição das seguintes informações do ambiente:

1- Obstáculo à oeste;
2- Obstáculo ao norte;
3- Obstáculo à leste;
4- Obstáculo ao sul.

E os atuadores permitem ao agente realizar os seguintes movimentos:

1- Deslocamento a oeste ou saída no sentido para a esquerda;
2- Deslocamento ao norte ou saída no sentido para cima;
3- Deslocamento a leste ou saída no sentido para a direita;
4- Deslocamento ao sul ou saída no sentido para baixo.

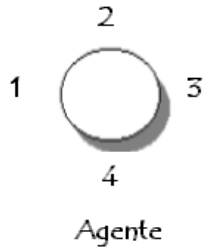

Figura 09: Representação do agente implementado, sensores e direções

A representação do ambiente constitui-se em uma matriz composta por zeros e uns, onde células com valores 0 são caminho livre e 1 são obstáculo. Construindo o mundo, como visto na figura 10 abaixo, a agente está representado pela letra “A”, as células em branco constituem caminho livre e as preenchidas com “X” representam obstáculo.

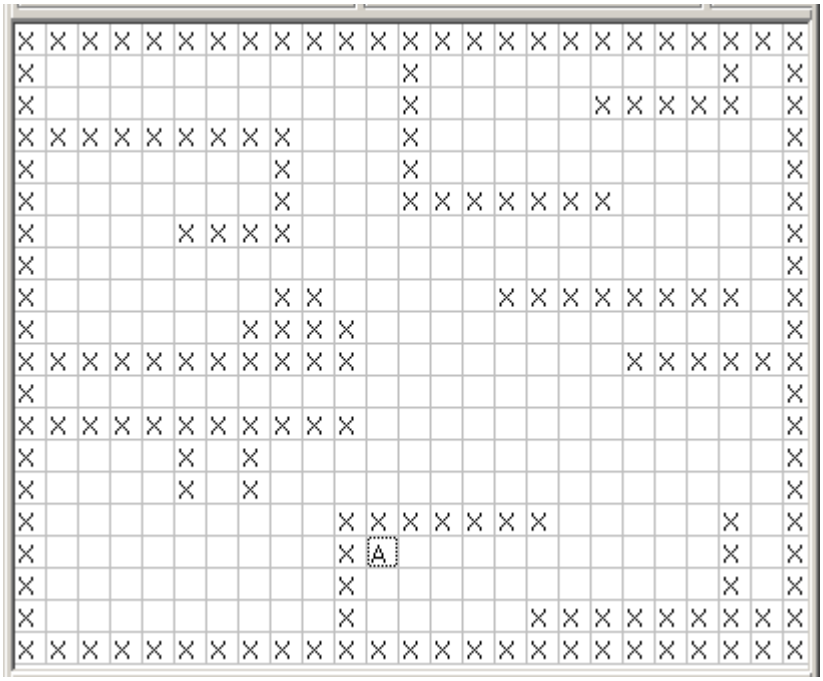

Figura 10 – Ambiente de simulação para teste da cognição do agente

Deste mesmo ponto de dados binários, os quatro sensores do agente, foram modelados, com entrada alta significando sensor habilitado ou presença de obstáculo e saída alta representando a direção a ser tomada pelo agente.

A tabela abaixo exemplifica as quinze situações válidas possíveis, sendo que a décima-sexta com ambos sensores habilitados figura como inexistente, já que excluiria totalmente a possibilidade do agente se mover, dado que está com obstáculos por todos os lados.

Situação	**Entradas**				**Saídas Permitidas, Dada a Entrada**			
	Sensor 1	Sensor 2	Sensor 3	Sensor 4	Oeste	Norte	Leste	Sul
01	0	0	0	0	1	0	0	0
					0	1	0	0
					0	0	1	0
					0	0	0	1
02	0	0	0	1	1	0	0	0
					0	1	0	0
					0	0	1	0
03	0	0	1	0	1	0	0	0
					0	1	0	0
					0	0	0	1
04	0	0	1	1	1	0	0	0
					0	1	0	0
05	0	1	0	0	1	0	0	0
					0	0	1	0
					0	0	0	1
06	0	1	0	1	1	0	0	0
					0	0	1	0
07	0	1	1	0	1	0	0	0
					0	0	0	1
08	0	1	1	1	1	0	0	0
09	1	0	0	0	0	1	0	0
					0	0	1	0
					0	0	0	1
10	1	0	0	1	0	1	0	0
					0	0	1	0
11	1	0	1	0	0	1	0	0

					0	0	0	1
12	1	0	1	1	0	1	0	0
13	1	1	0	0	0	0	1	0
					0	0	0	1
14	1	1	0	1	0	0	1	0
15	1	1	1	0	0	0	0	1
16	1	1	1	1	Inexistente (robô totalmente fechado)			

Tabela 01: Mapeamento das entradas dos sensores do agente em saídas permitidas

5.2 Proposta um: aprendizagem por reforço em uma única rede direta

5.2.1 Descrição da proposta

Um primeiro passo para a realização da implementação e testes foi a escolha da arquitetura de rede neural para aplicação da idéia de aprendizagem por reforço. Vários testes realizados em SOUZA & SILVA (2001) consideraram a rede direta como sendo capaz de implementar o conceito de aprendizagem por reforço. Além disso, a opção por redes neurais diretas constitui-se uma idéia tentadora, por ser uma arquitetura relativamente simples e portanto com facilidade de uso.

A implementação em primeira instância propôs então uma rede direta para ensinar ao robô em tempo real a sentir a presença de obstáculos como uma característica danosa ao seu objetivo de vagar pelo ambiente e que portanto estes deveriam ser evitados. Para isso, a cada colisão, o robô deveria ser punido ou ter um ou mais dos pesos da sua rede neural modificados.

Definida uma política de punição, partiu-se para escolha do método para se aplicar esta punição. SOUZA & SILVA (2001) propuseram um método de punição e recompensa que procura trazer as idéias clássicas de aprendizagem por reforço para redes neurais artificiais. Neste método, deve-se armazenar as informações sobre o estado anterior da rede e sua resposta para compará-lo com os estados atual e a nova resposta. Uma conexão e posteriormente um peso é escolhido de forma aleatória. Este peso é então alterado e obtêm-se uma nova resposta da rede. Se esta resposta resultar em um novo estado satisfatório deve-se manter a tendência de alteração deste peso e escolher outro aleatoriamente para alteração, repetindo o processo indefinidamente até que o novo estado não seja mais satisfatório. Neste caso retorna-se o valor do último peso alterado e seleciona-se outro aleatoriamente para alteração. Seguindo o processo até que um estado suficientemente satisfatório seja atingido.

5.2.2 Análise da Proposta

Embora os inúmeros testes com vários métodos híbridos apresentados em SOUZA & SILVA (2001) confirmassem a plausibilidade do uso de redes diretas para a implementação dos conceitos de aprendizagem por reforço, verificou-se que embora o agente aprendesse de fato uma determinada situação, ao enfrentar uma nova situação, este "esquecia" a aprendizagem construída na primeira situação. A figura 11 mostra duas situações de aprendizagem de situações pelo agente. Na situação X, o agente aprende a sair de uma situação com obstáculos a oeste e norte, mas quando parte para uma nova situação Y, com obstáculos a norte e leste, perde o conhecimento anterior adquirido na situação X. Assim, começou-se a pensar em algum tipo de arquitetura que provesse memória de situações aprendidas.

Situacão X

Situacão Y

Figura 11: Trajetória do agente evidenciando situações de aprendizagem diferentes

5.3 Proposta dois: Rede ART como habilitadora de várias redes diretas

A escolha pela arquitetura de rede neural ART se deve ao fato desta arquitetura armazenar padrões já conhecidos em categorias ou *clusters*, recuperando esse padrão quando já armazenado, permitindo uma aprendizagem plástica e estável, proporcionando uma espécie de memória de padrões. Neste caso específico, uma memória de situações (posições de obstáculos no mundo) encontrados pelo agente. Dentre os muitos tipos de ART existentes, optou-se por utilizar a rede ART1, que trabalha com dados binários, visto que a modelagem segue este formato de tratamento de dados para indicar existência de obstáculo, habilitação dos sensores e dos atuadores do agente.

A nova arquitetura prevê o uso concomitante de várias redes diretas aprendendo por reforço sendo controladas por uma rede ART1.

5.3.1 Descrição da arquitetura composta criada

A principal característica de uma rede ART é sua capacidade de criar nós de saída ou categorias, quando frente a padrões ainda não reconhecidos, sem esquecer os já armazenados. E desta forma, manter memória sobre padrões aprendidos.

Nesta etapa do trabalho propõe-se uma rede ART1 controlando várias redes diretas com aprendizagem por reforço. A rede ART1 fica responsável por armazenar como seus padrões o índice da rede direta que aprende por reforço a contornar os obstáculos do ambiente pelo agente, como pode ser visto na figura 12:

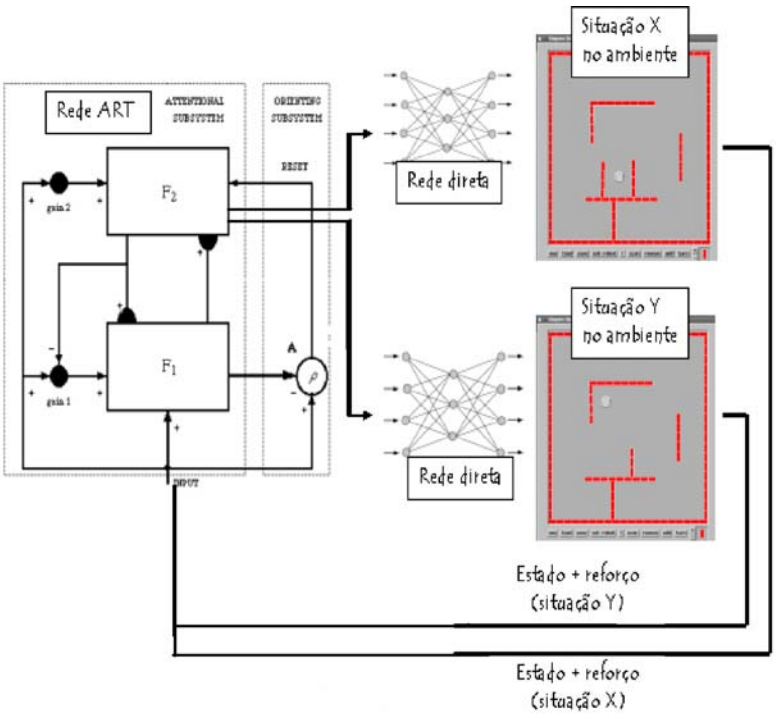

Figura 12: Visão geral da arquitetura proposta, interligando uma rede ART1 a várias redes diretas

Quando tratamos de um tipo de aprendizagem não-supervisionada com treinamento em tempo real, como acontece com a aprendizagem por reforço, estamos assumindo valores aleatórios de entrada (conjunto de exemplos), já que não dispomos de um conjunto de exemplos pré-treinados, como ocorre com nos casos de treinamento a priori.

Baseado nessa afirmação, realizou-se a etapa de testes em duas fases, em módulos separados e depois conectando-se as duas redes neurais. Primeiramente testou-se a rede

ART1 em separado para averiguação da maneira como esta se comporta no armazenamento de seus padrões de entrada e para seu melhor entendimento.

Partiu-se então para a construção dos dois módulos-pais da implementação deste trabalho, que a partir de agora serão chamados de módulo “ART” e módulo “DIRETA-COM-REFORÇO”, como vistos na figura 13.

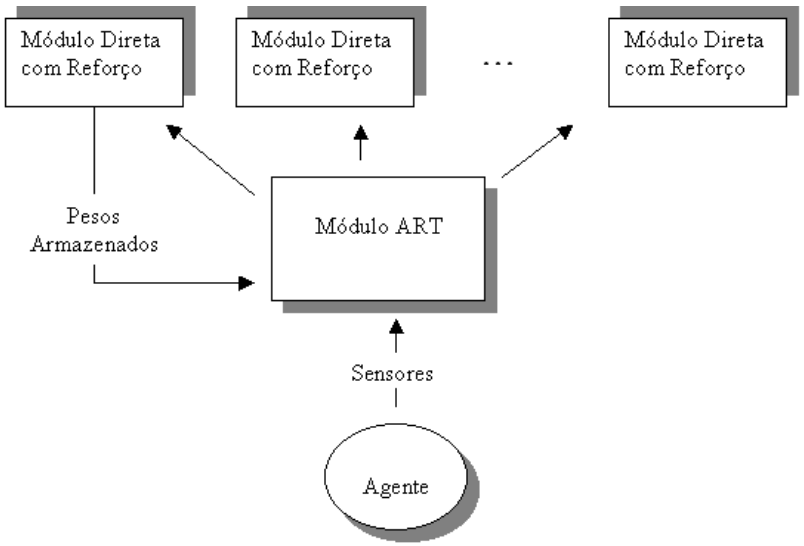

Figura 13: Diagramação dos módulos da arquitetura proposta

5.3.1.1 O Módulo "DIRETA-COM-REFORÇO"

Embora os módulos DIRETA-COM-REFORÇO sejam subseqüentes ao módulo ART no funcionamento da arquitetura, a implementação e os testes foram realizados de modo bottom-up e, portanto, é necessário que se comece o entendimento por esse módulo.

A primeira parte da implementação consistiu em definir a quantidade de neurônios na camada de entrada, na camada intermediária e na camada de saída. Para este módulo, primeiramente operando sozinho, em função de teste inicial, definiram-se quatro neurônios na camada de entrada, cada um representando um dos quatro sensores do agente, quatro

neurônios na camada intermediária e quatro neurônios na camada de saída, explicitando as possíveis direções que o agente poderia seguir.

Padronizadas entradas e saídas, ambas em formato binário, passou-se à aplicação do conceito de aprendizagem por reforço na rede direta. Para tal, definiu-se utilizar a política de punição ao agente, quando este tivesse um ou mais de seus sensores habilitado(s), ou seja, em processo de colisão com obstáculo. Esta punição se daria efetivamente na forma de se alterar um peso randomicamente. Para isso, implementou-se uma rotina de escolha aleatória primeiramente da conexão (camada de entrada para camada intemediária ou camada intermediária para camada de saída) e posteriormente do peso (sorteio de uma posição –linha e coluna- da matriz de pesos) a ser alterado (randomizado). Esta alteração está contida numa faixa de valores pequenos, mais especificamente [-0.009,0.009]. As informações sorteadas são armazenadas e a alteração é somada ao peso efetuando assim a modificação na rede. O algoritmo em pseudo-código ilustra esse processo:

```
Para ide 1 até m E  j de 1 a n faça
  Randomizar (Peso[i][j])   //Matriz de pesos da rede inicializada randomicamente
  Enquanto (;;) //Laço infinito, robô vagando pelo ambiente sem tempo demarcado
        Repita
          Se Habilita(sensor) ← verdadeiro //Se houver obstáculo em algum sensor
            Punição ←  verdadeiro          //Habilitar punição
              Escolher (conexão[k])
                    Escolher (Peso[i][j])
                    Peso[i][j] ← Randomizar(Peso[i][j]) //Alterar peso
                    S←Wij*Peso[i][j]  //Propagar peso alterado pela rede
             Punição←falso
```

```
                Até que Habilita(sensor) ←falso //Fim repita, célula sem obstáculo
        Fim_Enquanto
    Fim_Para
```

Os pesos da rede neural direta que controlam a interação do agente com o ambiente são inicializados com pesos aleatórios. O agente recebe as informações do ambiente (presença/ausência de obstáculo) e as repassa para a rede direta. A rede por sua vez, propaga os estímulos recebidos e determina o neurônio vencedor na saída, o qual indicará uma direção possível de ser trafegada pelo agente, isto é, sem obstáculo. Note-se entretanto que o agente não poderia seguir para duas direções ao mesmo tempo, assim, quando o agente tem por exemplo o vetor de saídas com mais de uma direção habilitada, nesse caso também recebe uma punição (alteração de um peso) até que seja gerado pela rede um vetor de saída com somente um neurônio com saída alta, sendo ainda esta posição deste vetor uma saída permitida ou não-igual a mesma posição alta do vetor de entrada, o que indica presença de obstáculo. Por recompensa, neste caso, entende-se que os pesos da rede neural direta permanecem inalterados ou que o agente está agindo de forma adequada no mundo.

Assim, a cada vez que o agente for punido, a rede vai sendo alterada em um peso de uma conexão, previamente sorteados, até que ela produza uma saída válida. A tabela 01 mostrada na seção anterior explicita em cada hachura uma situação diferente que pode ser encontrada pelo agente e as saídas válidas que podem ser aprendidas pela rede neural direta através do reforço por punição.

5.3.1.2 O Módulo ART

O segundo passo da implementação foi a construção do módulo ART, a rede que seria a controladora das diversas redes neurais diretas. A idéia parte do princípio de que a rede neural ART atua como memória do agente para as situações encontradas, já que sua principal característica é a capacidade de criar novas categorias para armazenamento de

padrões ainda não conhecidos, bem como a recuperação de um padrão, quando já conhecido. Assim, se preserva a utilização de redes diretas para aprendizagem de cada situação diferente encontrada pelo agente, mas com a idéia de se utilizar uma rede neural ART funcionando como um *switch* neural, ou seja, um chaveador das várias redes diretas, habilitando cada rede que resolvesse uma determinada situação de obstáculo e escalonando essa rede, quando fosse preciso, em outro momento de encontro de obstáculo pelo agente em posição idêntica.

Assim os quatro sensores de entrada do robô passam a ser o vetor de entrada da rede ART1. Para exemplificar, segue-se um roteiro de como o módulo ART1 funcionaria em um processo de reconhecimento de padrões, a partir do recebimento do vetor de entrada dos sensores.

A aprendizagem em uma rede ART se dá através de modificadores de pesos ou traços LTM. Esse processo de aprendizagem não começa nem termina, mas continua inclusive enquanto ocorre o processo de "*matching* " (busca por padrões existentes). Cada vez que se enviam sinais através das conexões, os pesos associados a eles sofrem modificações. Essa característica faz alusão à filosofia de aprendizagem em tempo real proporcionada pela aprendizagem por reforço. Não se percebem fases definidas de treinamento e aprendizagem. As modificações dos pesos são percebidas inclusive simultaneamente durante a fase de associação dos padrões ou *matching*.

Levando-se em conta essa dinâmica de aprendizagem, há que se perguntar se não há perda de conhecimento ou aprendizagem de associações incorretas. Em resposta a esses questionamentos FREEMAN&SKAPURA (1991) afirma que o tempo necessário para que se produzam mudanças significativas nos pesos é muito grande em relação ao tempo necessário para um ciclo completo de busca de "*matching*". As conexões que participam nas buscas faltosas não estão ativas em tempo suficientemente grande para que os pesos associados sejam afetados gravemente.

Quando se produz uma associação, não há sinal de restauração ("*reset*") e a rede se estabelece em um estado ressonante. Durante este estado estável, as conexões permanecem ativas por um tempo suficientemente longo para que os pesos sejam fortalecidos.Este estado ressonante só pode surgir quando se produzir um reconhecimento de padrões ou durante o surgimento de novas unidades de F2 para armazenar um padrão desconhecido previamente. Cada unidade de F2 nesta implementação se constitui em um índice que identifica unicamente uma rede neural direta que aprendeu por punição a resolver uma dada situação encontrada.

Seguindo-se esta filosofia, a rede neural que aprendeu cada situação de obstáculo e resolveu através de uma saída válida deve ser recuperada a cada vez que o robô se deparar com o mesmo padrão de entrada de seus sensores. Por saída válida entenda-se sem obstáculo e fazendo com que o robô atinja uma única direção, visto que não poderia mover motores contrários de uma só vez. Importante ressaltar que cada rede não sofrerá mais o retorno dos valores alterados em estados anteriores, sendo que a cada punição, os pesos formarão um novo *cluster* (ou categoria) na rede neural ART, alocando uma rede neural para cada situação possível.

5.3.2 Exemplificação do funcionamento da arquitetura proposta

Esta seção exemplifica a execução da arquitetura proposta em passos. Supondo um mundo como o da figura abaixo, e o agente encontrando-se na posição indicada pela letra "A" tem-se a seguinte situação: o agente encontra-se em uma posição com obstáculo à esquerda e à direita. Esta interpretação do estado é fundamentalmente a entrada para o módulo ART. Baseada na entrada, a ART, alocando uma rede neural direta, ou seja, passa estas entradas para ela. A rede direta, por sua vez, gera uma saída que vai ser alterada (agente punido) até que produza uma saída válida, que neste caso configura saída a norte ou para cima (0100) ou saída a sul ou para baixo (0001). A partir do momento em que esta rede direta produzir uma saída possível, ela movimentará o agente para a direção válida e

seu índice será armazenado como uma nova categoria da rede ART. Assim, da próxima vez que o agente passar por uma situação semelhante, a rede ART habilitará a rede direta que aprendeu a resolver este problema.

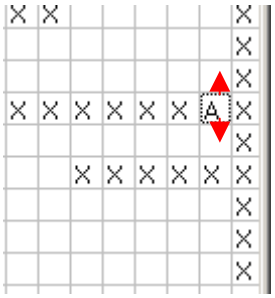

Figura14: Situação específica encontrada pelo agente no mundo criado com representação das saídas possíveis

5.3.3 Análise da arquitetura proposta

Concebido o modelo de arquitetura neural que resolvesse as limitações de memória de ações do agente no ambiente, partiram-se para os teste da arquitetura proposta. Esta foi testada inicialmente com vários formatos de mundo diferentes, propondo obstáculos para o agente em várias posições diferentes e concomitantes.

Para a aprendizagem do comportamento por reforço, escolheu-se primeiramente utilizar a política de punição do agente a cada colisão através da alteração de um peso aleatório da rede direta que estaria aprendendo o comportamento em tempo real. Esse método de punição imediata apesar de funcionar, no sentido de deslocar o agente para um próximo estado válido, apresentou o problema de o agente ficar preso em obstáculos simétricos muito próximos, ou seja, em um laço de repetição de movimentos. A rede ART já tendo armazenado as redes diretas para cada situação, alocaria estas redes infinitamente, ou seja, deslocando o agente sempre pra mesma posição. Um exemplo pode ser notado na figura 15, onde o agente sendo punido no estado B seria mandado ao estado C e no sendo

punido no estado C seria deslocado novamente para o estado B e assim sucessivamente, configurando um laço de repetição de mesmos movimentos.

Estado B Estado C

Figura 15: Laço gerado pela recompensa imediata nos estados B e C

Como forma de resolver tal restrição, utilizou-se a noção de recompensa postergada, a qual implementa a idéia de se fazer uma média de punições recebidas em um número variável predefinido de iterações ou estados sentidos pelo agente. Desta forma, o agente só será punido se agir muitas vezes da mesma forma numa quantidade estipulada de vezes, aumentando a probabilidade de que a rede ART resolva a mesma situação com outra rede direta que desloque o agente para um novo estado. Por exemplo, movendo o agente no estado B para próxima célula a sul ou norte, evitaria o looping gerado se o agente fosse movido para a próxima célula a leste.

Testes foram feitos com números variáveis de passos (dez, cinqüenta e cem), não apresentando, entretanto, grande melhora em detrimento do tempo maior gasto.

Em relação à política usada na aprendizagem em tempo real, ou seja, punição do agente através da alteração de um único peso aleatório, testou-se uma nova forma de punição, alterando-se (randomizando-se) todos os pesos da conexão escolhida. Este novo tipo de punição, mais drástica, surtiu melhor efeito na verificação da aprendizagem de cada situação diferente, isto é, de cada rede direta responsável por aprender cada situação de

obstáculo. O agente com esse tipo de punição demonstrou melhor desempenho na locomoção de um estado para outro no ambiente.

5.4 Proposta três: Rede ART realimentada com neurônios de estado

5.4.1 Descrição

Como forma de confirmar a eficácia da arquitetura híbrida proposta, submeteu-se o agente ao teste do labirinto. O mundo foi reconstruído como um labirinto em formato de S e colocou-se um objetivo ao fim do caminho, forçando o agente a se deslocar até este, como pode ser visto na figura 16. Como se pode notar, do ponto de vista do agente, os estados B e C são iguais. Logo, a tendência da arquitetura proposta é resolver o problema com a mesma rede direta. O que se espera, do ponto de vista do observador, é que no estado B, o agente aprenda a mover-se para o sentido Oeste e no estado C, aprenda a mover-se para o sentido Leste. Para que a rede ART consiga alocar diferentes redes diretas pra situações iguais, torna-se necessário para o agente o armazenamento do estado anterior ao atual.

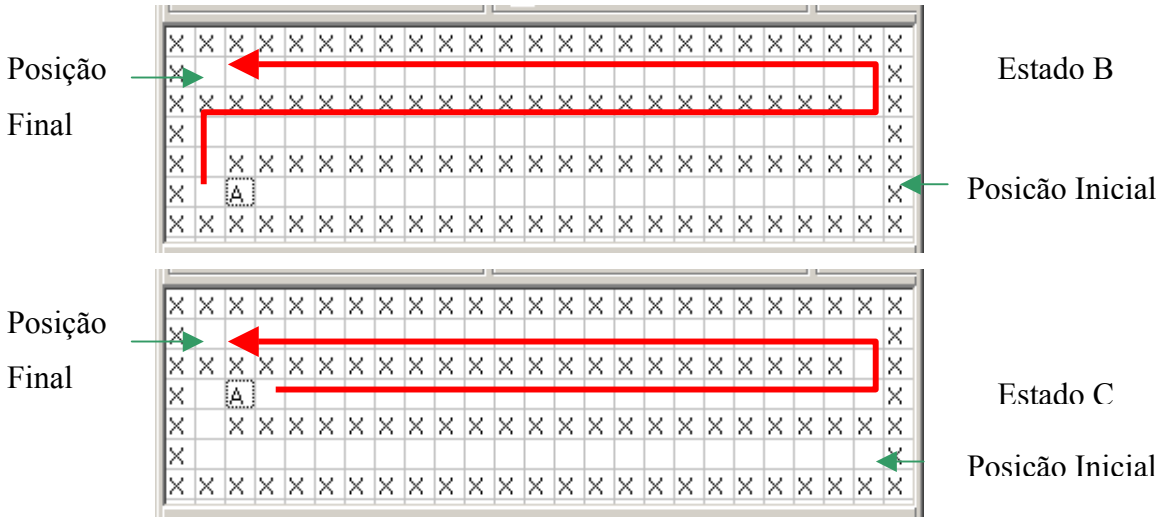

Figura 16: Labirinto reproduzindo uma versão mais complexa de ambiente com dois estados

5.4.2 Análise da terceira proposta

Com a implementação dos neurônios adicionais identificadores do estado anterior, criou-se a idéia de recorrência de aprendizagem de estados e foi possível melhorar a capacidade de discriminação de estados. Para tal, introduziram-se mais quatro neurônios na camada de entrada da rede ART. Assim sendo, os quatro primeiros identificariam as entradas anteriores a que o agente se submeteu e os quatro neurônios subseqüentes continuariam sendo responsáveis pela estado atual em que o agente se encontra.

Com esta nova abordagem, torna-se possível a diferenciação entre estado anterior e estado atual do ponto de vista do agente, permitindo que este possa tratar situações aparentemente idênticas, como as vistas na figura 14, com resoluções não necessariamente iguais, isto é, movimentar-se para direções diferentes em uma situação já aprendida, redefinindo seu aprendizado, conforme a necessidade, como no caso de um mundo mais complexo.

A figura 17, a seguir, esboça a terceira arquitetura proposta composta de uma rede neural ART com realimentação de estados habilitando redes diretas. Entendam-se os neurônios das camadas F1 e F2 da rede ART completamente interconectados. A camada F1 da ART não só possui os quatro neurônios responsáveis pela leitura dos sensores de orientação (Norte, Sul, Leste, Oeste), como agora também apresenta mais quatro neurônios denominados na figura de DN (direção norte), DS (direção sul), DL (direção leste) e DO (direção oeste). Após a camada F2 alocar uma determinada rede direta para aprender uma situação por reforço (através do neurônio habilitador setado como nh), os neurônios de saída desta rede direta realimentam as quatro novas entradas da camada F1 da rede ART, que serão responsáveis por informar a rede ART estado anterior vivido pelo agente, isto é, os atuadores habilitados no estado anterior.

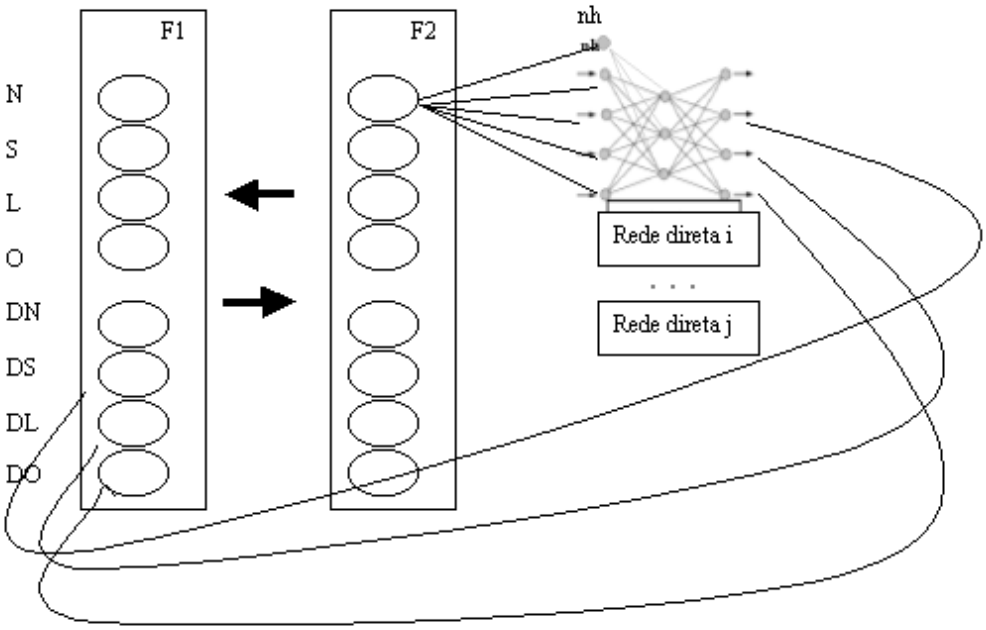

Figura 17: Esboço da terceira proposta de implementação

6 Conclusões e Trabalhos Futuros

Em se tratando de aprendizagem de robôs baseados em comportamento, onde o paralelismo dos comportamentos independentes é uma preocupação inerente, não é raro se pensar na utilização de redes neurais artificiais como meio.

A propriedade mais importante das redes neurais é a habilidade de aprender de seu ambiente através de exemplos e generalizar esta aprendizagem de maneira a reconhecer instâncias similares, visando uma melhora de desempenho.

Dentre os vários tipos de redes neurais, a opção pela utilização de uma rede ART se deve ao fato deste tipo manter equilíbrio entre as propriedades de plasticidade (discriminação) e de elasticidade (generalização). Ou seja, capacidade de criar uma nova categoria de padrões, quando ocorrer estimulação por padrões não reconhecidos e ainda agrupar padrões similares na mesma categoria, quando reconhecidos. Dessa forma, a rede responde rapidamente a dados aprendidos previamente e ainda é capaz de aprender quando novos dados são apresentados. Além disso, modelos ART são mais adaptáveis para aprender e responder em tempo real, para um mundo não-estacionário.

O uso da rede ART tem a capacidade de resolver o dilema da plasticidade/elasticidade, descrito no capítulo três desta dissertação e de se constituir em uma rede que suporta aprendizagem não-supervisionada, característica desejada frente ao intuito de se aplicar aprendizagem por reforço em sistemas robóticos móveis autônomos. Outra característica que chama a atenção para o uso deste tipo de rede ser apropriado é a propriedade da rede ART de apresentar aprendizagem incremental e plástica, segundo SALOMON(1996), isto é, determinadas ações associadas com sensores podem mudar.

Quanto às redes diretas para a aprendizagem das situações do agente, a escolha se deu pela facilidade apresentada por este tipo de topologia em relação a seu uso e sua implementação

quando aplicadas à aprendizagem por reforço. A aplicação da arquitetura híbrida de uma rede neural ART escalonando redes diretas é uma tentativa de assegurar memória à aprendizagem em tempo real proporcionado pela aprendizagem por reforço.

Em um contexto neurobiológico, memória se refere às alterações neurais relativamente duradouras induzidas pela interação de um organismo com o seu ambiente e para que seja útil, ela deve ser acessível ao sistema nervoso, para poder influenciar o comportamento futuro, segundo TEYLER(1986) apud HAYKIN(1999). HAYKIN(1999) cita ainda que um padrão de aprendizagem deve ser armazenado na memória através de um processo de aprendizagem. Quando um padrão de atividade particular é aprendido, ele é armazenado no cérebro, de onde pode ser recuperado mais tarde, quando exigido. O termo aprendizagem está intimamente ao conceito de memória. Este trabalho teve como foco de verificação de sucesso a memória situações anteriormente aprendidas. O agente mostrou-se capaz de reconhecer situações aprendidas anteriormente e de executar a transposição destas situações da mesma maneira, como quando do período de aprendizagem.

Com o esforço de trazer os conceitos clássicos da aprendizagem por reforço à luz de um paradigma conexionista, e com os diversos testes realizados em várias abordagens de arquiteturas de redes neurais artificiais pretendeu-se provar, mesmo sendo métodos não-convencionais, a relativa facilidade e comprovada eficiência da utilização de redes neurais na aprendizagem inteligente em tempo de operação de um comportamento com memória de situações por um agente autônomo adaptativo.

6.1 Perspectivas futuras a partir deste trabalho

A inspiração biológica tem sido a fundamentação desde o conceito de redes neurais artificiais até as mais variadas aplicações envolvendo esse paradigma de programação

paralela. A idéia da aprendizagem de comportamentos em agentes autônomos a partir de redes neurais não fugiu a esta regra. Assim sendo, há muito o que se fazer ainda, tendo a etologia como base. Este trabalho propõe uma arquitetura híbrida composta de duas topologias diferentes para explicitar memória de um único comportamento.

Não há como não questionar uma possível hierarquia de comportamentos independentes e em funcionando em paralelo com memória de ações e situações e aprendizagem em tempo real.

Outra sugestão de pesquisa futura inclui outras formas de arquiteturas híbridas, propondo o uso de topologias combinadas. Redes ART combinadas a redes recorrentes ou ainda duas redes ART, cada uma controlando a memória de situações de cada comportamento emergente do agente, bem como o estudo de ambientes mais complexos com estas redes.

Existem poucos estudos sobre agentes colaborativos utilizando aprendizagem por reforço em mundos não-estacionários. Aspectos interessantes poderiam surgir de uma arquitetura que guardasse a memória de interação entre esses agentes.

Quanto à arquitetura proposta, um ponto importante de estudo futuro pode concentrar o uso de diferentes redes ART como alternativa de aprendizagem de comportamento com memória. Talvez a criação hipotética de uma RART (Reinforcement ART), uma rede ART com reforço embutido direto na camada de reconhecimento (F2), com possibilidade de decrementar ou incrementar funções de vigilância, como punição ou recompensa.

7 Referências Bibliográficas

(ANDERSON et al 1990) ANDERSON, T. L., DONATH,M. Animal behavior as a paradigm for developing robot autonomy. In P. Maes, editor, Designing Autonomous Agents: Theory and Practice from Biology to Engineering and Back, A Bradford Book, p. 145--168. The MIT Press, Cambridge, Massachusetts, 1990.

(ARKIN 1998) ARKIN, R.C. Behavior-based Robotics. MIT Press, Cambridge, Massachusetts, USA: 1998.

(ART CLEARINGHOUSE 2002). ART1 CLEARINGHOUSE Repositório de Redes Neurais ART1. Disponível em http://www.cs.umr.edu/ART1. Última atualização feita em setembro de 2002. Acesso em dezembro de 2002.

(BARRETO 2001) BARRETO, J. M. Inteligência Artificial no Limiar do Século XXI. Ed. Duplic. Florianópolis: 1999.

(BEER et al 1990) BEER, R. D., CHIEL, H. J. and STERLING, L. S. A biological perspective on autonomous agent design. In P. Maes, editor, Designing Autonomous Agents: Theory and Practice from Biology to Engineering and Back, A Bradford Book, p.169--186. The MIT Press, Cambridge, Massachusetts, 1990.

(BITTENCOURT 1998) BITTENCOURT, G. Inteligência Artificial. Ferramentas e teorias. Ed. UFSC. Floranópolis: 1998

(BROOKS 1986) BROOKS, R. A Robust Layered Control System for a Mobile Robot. IEEE Journal of Robotics and Automation, 2 (1), 1986.

(BROOKS 1989) BROOKS, R. A. A robot that walks; emergent behaviors from a carefully evolved network. A. I. Memo 1091, Massachusetts Institute of Technology, Artificial Inteligence Laboratory, Feb. 1989.

(BROOKS 1990a) BROOKS, R. Elephants don't play chess. Designing Autonomous Agents. Theory and Practice from Biology to Engineering and back. MIT Press. Cambridge, 1990.

(BROOKS 1990b) BROOKS, R. The Behavior Language; User's Guide. MIT Press. Cambridge, Massachussets, USA. 1990.

(BROOKS 1991) BROOKS, R. A. Artificial life and real robots. In F. J. Varela and P. Bourgine, editors, Proc. of the First European Conference on Artificial Life, A Bradford Book, p. 3--10, Paris, Dec. 1991. The MIT Press.

(BROOKS&MAES 1990). BROOKS, R., MAES, P. Learning to Coordinate Behaviors. In: Proceedings of the 8th Nacional Conference on Artificial Intelligence.____: 1990.

(CARPENTER et al 1991) CARPENTER,G.A., GROSSBERG,S., REYNOLDS,J.H. ARTMAP: Supervised Real_time Learning and Classification of Nonstationary Data by a Self-Organizing Neural Network. In: Neural Networks, Vol.4, pg. 565-588. USA, 1991

(FREEMAN & SKAPURA 1991) FREEMAN,J.A., SKAPURA, D.M. Neural Networks: Algorithms, Aplications and Programming Techniques. Ed. Addison-Wesley Publishing Company. 1992

(GUAZELLI 1994) GUAZELLI, Alex. Aprendizagem em sistemas híbridos. Dissertação de mestrado apresentada ao programa de pós-graduação em Ciência da Computação da Universidade Federal do Rio Grande do Sul. Porto Alegre: 1994.

(HAYKIN 1994) HAYKIN, S., Neural Networks - A Comprehensive Foundation, Macmillan Publishing Company, Inc. ,1994.

(HAYKIN 2001) HAYKIN, S. Redes Neurais. Princípios e Prática. Bookman. Porto Alegre:2001

(KAELBLING et al 1996) KAELBLING, L., LITTMAN, M., MOORE, A. Reinforcement Learning: a Survey. Disponível em: http://www2.cs.cmu.edu/afs/cs/project/jair/pub/volume4/kaelbling96a-html/rl-survey.html. Acesso em outubro de 2001.

(KOVÁCS 1996) KOVÁCS, Z. L. Redes Neurais Artificiais: Fundamentos e Aplicações. Collegium Cognitio. São Paulo. 1996

(LEITE et al 1993) LEITE, A., ALVES, J. B. da M. e RESENDE, M. F. Sistema para depuração de um robô móvel autônomo inteligente. In Anais do 1. Simpósio Brasileiro de Automação Inteligente, p. 175--184, Rio Claro, SP, 1993.

(LIN 1991) LIN, L. Programming robots using reinforcement learning and teaching. In Proc. of the Ninth National (USA) Conference on Artificial Intelligence, p. 781--786, Menlo Park, CA, July 1991. The MIT Press.

(LOESCH&SARI 1996) LOESCH, C. , SARI, S.T. Redes Neurais Artificiais: Fundamentos e Modelos. Ed. Da FURB. Blumenau: 1996.

(MAES 1990) MAES, P. Modeling Adaptive Autonomous Agents. MIT Press. Cambridge, 1990.

(MAES 1991) MAES, P. Learning behavior networks from experience. In F. J. Varela and P. Bourgine, editors, Proc. of the First European Conference on Artificial Life, A Bradford Book, p. 48--58, Paris, Dec. 1991. The MIT Press.

(MAHADEVAN&CONNEL 1991) MAHADEVAN, S.,CONNEL, J. Automatic Programming of Behavior-based Robots using Reinforcement Learning. In: Proceedings of the 8th Nacional Conference on Artificial Intelligence. Anaheim, CA: 1991

(MATARIC 1997) MATARIC, M. Behavior-Based Control: Examples from Navigation, Learning, and Group Behavior. Waltham, MA: 1997

(MATARIC 1998) MATARIC, M. A History-Based Approach for Adaptive Robot Behavior in Dynamic Environments. Proceedings in Second International Conference on autonomous agents. Minneapolis: 1998.

(McFARLAND & BÖSSER 1993) McFARLAND, D.,BÖSSER, T. Intelligent Behavior in Animals and Robots. MIT Press. Cambridge,MA: 1993.

(MITCHEL 1997) MITCHEL, T. Machine Learning. McGraw-Hill. USA, 1997.

(OLIVEIRA 2001) OLIVEIRA, L. O. Mapas auto-organizáveis de Kohonen aplicados ao mapeamento de ambientes de Robótica Móvel. Dissertação de mestrado submetida ao

Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Santa Catarina. Florianópolis: 2001.

(ROISENBERG 1996) ROISENBERG, M. Emergência da Inteligência em Agentes Autônomos através de Modelos Inspirados na Natureza. Tese de Doutorado. Departamento de Engenharia Elétrica, Grupo de Pesquisas em Engenharia Biomédica, UFSC. Florianópolis: 1998.

(RUSSELL &NORVIG 1995) RUSSELL,S., NORVIG,P. Artificial Intelligence: A Modern Approach, Prentice-Hall, 1995

(SALOMON 1996) SALOMON,R. Neural Networks in the Context of Autonomous Agents: Important Concepts Revisited. In: Proceedings of the Artificial Neural Networks in Engeneering. New York, 1996.

(SILVA 2001) SILVA, F.A. Redes Neurais Hierárquicas para Implementação de Comportamentos em Agentes Autônomos. Dissertação de mestrado submetida ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Santa Catarina. Florianópolis: 2001.

(SOUZA & SILVA 2001) Souza, M. , Silva T. Redes Neurais com Aprendizado por Reforço: Estudos e Aplicações. Monografia de conclusão de curso submetida ao Intituto de Informática e Estatística da Universidade Federal de Santa Catarina. Florianópolis, 2001.

(SPIER 1997) SPIER, E. From Reactive Behaviour to Adaptive Behaviour. Tese de pós doutorado submetida a Universidade de Oxford. Trinity: 1997

(SUTTON 1998) SUTTON, R. Homepage do autor. Disponível em: http://www-anw.cs.umass.edu/~rich/sutton.html. Acesso em junho de 2002.

(SUTTON&BARTO 1998). SUTTON, R., BARTO,…Reinforcement Learning: an introduction. Livro virtual. Disponível em http://www-anw.cs.umass.edu/~rich/book/the-book.html. Acesso em junho de 2002.

(TAFNER 1998) TAFNER, M. As Redes Neurais Artificiais: Aprendizado e Plasticidade Disponível em: http://www.epub.org.br/cm/n05/tecnologia/plasticidade2.html. Copyright 1998. Acesso em julho de 2003.

(VALDIVIA2002). Pagina sobre Reinforcement Learning. Disponível em http://yvantv.tsx.org/. Acesso em dezembro de 2002.

(VALLE FILHO et. al 1997) VALLE FILHO, Adhemar Maria Do, CARRARO, José Luis, BASTOS, Lia Caetano, PEZZI, Silvana, ROMÃO, Wesley. Uma visão geral sobre rede neural artificial com arquitetura ART12. Revista Tecnológica, Maringá, v.6, 1997.

(VIEIRA 1999) VIEIRA, R. S. Protótipo de um sistema de monitoramento remoto inteligente. Dissertação de mestrado submetida ao Programa de Pós-Graduação em Engenharia de Produção da Universidade Federal de Santa Catarina. Florianópolis: 1999.

(WHITEHEAD&LIN 1996) WHITEHEAD, Steven D.; LIN, Long-Ji. Reinforcement learning of non-Markov decision processes. In: Computacional Theories of Interaction and Agency, Agre, P.E; Rosenchein,S. MIT, 1996

(ZALARNA 2002) ZALARNA, E. et. al. Adaptive Behavior Navigation of a Mobile Robot. In: Proceedings of IEEE Transactions on Systems, Man and Cybernetics. Vol. 32. Nº 1. 2002.

Anexo –

Código da Implementação em Linguagem de Programação C++

```
/*-------------------------------[]
Este programa trata a punição do
agente em curto e longo prazo.

A curto prazo a punição se da pelas situações:

1-A MLP não acende nenhum neurônio de saída
2-A MLP acende mais de um neurônio de saída
3-A MLP manda o agente se mover para onde tem um obstáculo


A longo prazo a punição se da pela situação:

1- Se agente repetiu muito seu comportamento, diante
de um numero máximo de iterações.
[]-------------------------------*/

/*****************************************************************
*******Declarações
*****************************************************************
******#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <conio.h>
#include <dos.h>
#include <windows.h>
```

```
typedef int           BOOL;
typedef char          CHAR;
typedef int           INT;
typedef double        REAL;

#define FALSE         0
#define TRUE          1
#define NOT           !
#define AND           &&
#define OR            ||


#define MIN_REAL      -HUGE_VAL
#define MAX_REAL      +HUGE_VAL

#define BIAS          1

typedef struct {                  /* A LAYER OF A NET:                    */
INT           Units;         /* - number of units in this layer       */
REAL*         Output;        /* - output of ith unit                  */
REAL*         Error;         /* - error term of ith unit              */
REAL**        Weight;        /* - connection weights to ith unit      */
REAL**        WeightSave;    /* - saved weights for stopped training  */
REAL**        dWeight;       /* - last weight deltas for momentum     */
} LAYER;
typedef struct {                  /* A NET:                               */
LAYER**       Layer;         /* - layers of this net                  */
LAYER*        InputLayer;    /* - input layer                         */
```

```
    LAYER*        OutputLayer;   /* - output layer                    */
REAL          Alpha;         /* - momentum factor                 */
REAL          Eta;           /* - learning rate                   */
REAL          Gain;          /* - gain of sigmoid function        */
REAL          Error;         /* - total net error                 */
} NET;


typedef struct {                  /* A LAYER OF A NET:              */
INT           Units;         /* - number of units in this layer   */
BOOL*         Output;        /* - output of ith unit              */
REAL**        Weight;        /* - connection weights to ith unit  */
BOOL*         Inhibited;     /* - inhibition status of ith F2 unit */
} LAYER_Art;

typedef struct {                  /* A NET:                         */
LAYER_Art*        F1;            /* - F1 layer                    */
LAYER_Art*        F2;            /* - F2 layer                    */
INT           Winner;        /* - last winner in F2 layer         */
REAL          A1;            /* - A parameter for F1 layer        */
REAL          B1;            /* - B parameter for F1 layer        */
REAL          C1;            /* - C parameter for F1 layer        */
REAL          D1;            /* - D parameter for F1 layer        */
REAL          L;             /* - L parameter for net             */
REAL          Rho;           /* - vigilance parameter             */
} NET_Art;

/*-----------------------------------------------------------------[]
```

```
    A P P L I C A T I O N - S P E C I F I C   C O D E
[]------------------------------------------------------------------*/
#define NUM_LAYERS    3        /*numero de camadas*/
#define N             4       /*unidades na camada de entrada*/
#define M             4        /*unidades na camada de saida*/

#define NUM_MAX_Populacao  15  /*Qtde total de RNAs na populacao*/

#define false 0
#define true  1

typedef struct Agente {INT x; INT y; }AgentPosition;
typedef struct Direcao {INT esq; INT cima; INT dir; INT baixo; }AgentDirecao;



AgentPosition Agent;
AgentDirecao ProxEstado;
                                                /*Esta segunda posicao do vetor de
unidades
                                                        eh  o  Numero
de Neuronios Intermediarios da RN*/
INT     Units[NUM_MAX_Populacao][NUM_LAYERS];


/*****************************************************************
************
                A P P L I C A T I O N - S P E C I F I C   C O D E
```

```
     ****************************************************************
***************/
#define N_Art            5
#define M_Art            15

#define NO_WINNER     M_Art

#define NUM_DATA      15

#define Iteracoes_Maximas  26

BOOL  Input_Art[NUM_DATA][N_Art]=
{
            {0,0,0,0,1},
            {0,0,0,1,1},
            {0,0,1,0,1},
            {0,0,1,1,1},
            {0,1,0,0,1},
            {0,1,0,1,1},
            {0,1,1,0,1},
            {0,1,1,1,1},
            {1,0,0,0,1},
            {1,0,0,1,1},
            {1,0,1,0,1},
            {1,0,1,1,1},
            {1,1,0,0,1},
            {1,1,0,1,1},
            {1,1,1,0,1}};
```

```
      BOOL  Output[M_Art];
INT   situacao_index; //Guarda a situacao do ambiente
INT   ClasseSaida;   //A classe determinada pela rede ART


//linha, coluna e indice da MLP
INT   Guarda_Posicoes_Agente[Iteracoes_Maximas][3];
INT   Iteracoes = 0;


/*-------------------------------------[]
Variaveis MLP - INICIO
[]-------------------------------------*/

/*Para rede MLP com backpropagation*/
NET  *Net[NUM_MAX_Populacao];

//Definição do ambiente : 0 - célula livre / 1 - obstáculo
INT Ambiente[25][25]= {
{1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1},
{1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,1,0,1},
{1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,1,1,1,1,1,0,1},
{1,1,1,1,1,1,1,1,1,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,1},
{1,0,0,0,0,0,0,0,1,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,1},
{1,0,0,0,0,0,0,0,1,0,0,0,1,1,1,1,1,1,1,0,0,0,0,0,1},
{1,0,0,0,0,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
{1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
{1,0,0,0,0,0,0,0,1,1,0,0,0,0,0,1,1,1,1,1,1,1,1,0,1},
```

```
      {1,0,0,0,0,0,0,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1},
{1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,1,1,1,1,1,1},
{1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
{1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
{1,0,0,0,0,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
{1,0,0,0,0,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
{1,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
{1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
{1,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,0,0,0,0,0,1,0,1},
{1,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,1,0,1},
{1,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,1,0,1},
{1,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,1,1,1,1,1,1,1,1,1},
{1,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
{1,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
{1,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,1},
{1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1}
};

//15 Situações de posição de obstáculo - ver tabela 01
REAL  Situacoes [15][4] = {
            {0,0,0,0},
            {0,0,0,1},
            {0,0,1,0},
            {0,0,1,1},
            {0,1,0,0},
            {0,1,0,1},
            {0,1,1,0},
            {0,1,1,1},
```

```
                {1,0,0,0},
            {1,0,0,1},
            {1,0,1,0},
            {1,0,1,1},
            {1,1,0,0},
            {1,1,0,1},
            {1,1,1,0}};


/*-------------------------------------[]
Variaveis MLP - FIM
[]-------------------------------------*/

BOOL Repeticao_Comportamento_Agente(void)
{
INT elemento, quant=1;

//Analisando as Colunas
elemento = Guarda_Posicoes_Agente[0][0];//tratando coluna
for (int i=1; i < Iteracoes_Maximas; i++)
{
if(elemento == Guarda_Posicoes_Agente[i][0])
quant++;
}
//média de iterações para punição postergada
if(quant >= 0.2*Iteracoes_Maximas) return(true);

quant = 1;
```

```
//Analisando as Linhas
elemento = Guarda_Posicoes_Agente[0][1];//tratando linha
for (int i=1; i < Iteracoes_Maximas; i++)
{
if(elemento == Guarda_Posicoes_Agente[i][1])
quant++;
}
if(quant == Iteracoes_Maximas) return(true);
else return(false);

}//fim funcao



/*-------------------------------------------[]
[]-------------------------------------------*/
INT RandomEqualINT(INT Low, INT High)
{
return rand() % (High-Low+1) + Low;
}
REAL RandomEqualREAL(REAL Low, REAL High)
{
return ((REAL) rand() / RAND_MAX) * (High-Low) + Low;
}


void InitializeApplication_Art(NET_Art* Net_Art)
```

```
    {
INT n,i,j;

//parâmetros de inicialização da Rede ART
Net_Art->A1  = 1;
Net_Art->B1  = 1.5;
Net_Art->C1  = 5;
Net_Art->D1  = 0.9;
Net_Art->L   = 3;
Net_Art->Rho = 0.9;

for (i=0; i<Net_Art->F1->Units; i++) {
for (j=0; j<Net_Art->F2->Units; j++) {
Net_Art->F1->Weight[i][j] = (Net_Art->B1 - 1) / Net_Art->D1 + 0.2;
Net_Art->F2->Weight[j][i] = Net_Art->L / (Net_Art->L - 1 + N_Art) - 0.1;
}
}
}

/*-----------------------------------------[]
Procedimento para randomizar pesos
[]-----------------------------------------*/
void RandomWeights(INT ID_Rede)
{
INT l,i,j;

for (l=1; l < NUM_LAYERS; l++) {
for (i=1; i <= Net[ID_Rede]->Layer[l]->Units; i++) {
```

```
      for (j=0; j <= Net[ID_Rede]->Layer[l-1]->Units; j++) {
Net[ID_Rede]->Layer[l]->Weight[i][j] = RandomEqualREAL(-1, 1);
}
}
}
}

//ART interpreta entrada ambiente (posição do obstáculo)
void WriteInput_Art(BOOL* Input)
{
INT i;
gotoxy(30,10);
printf("Situacao do Ambiente: ");
for (i=0; i < N_Art-1; i++) {
printf("%c", (Input[i]) ? '1' : '0');
}
//getch();
}

//Saída da Rede ART - rede direta escolhida
void WriteOutput_Art(NET_Art* Net_Art)
{
if (Net_Art->Winner != NO_WINNER)
{
gotoxy(30,13);
printf("A Rede ART escolheu: MLP %i  ", Net_Art->Winner);
//getch();
```

```
      ClasseSaida = Net_Art->Winner;
}
else
{ gotoxy(30,12);
printf("new Input and all Classes exhausted!");
getch();
gotoxy(30,12);
printf("                        ");

}
}
/*****************************************************************
************
Procedimento para gerar rede ART
        I N I T I A L I Z A T I O N
*****************************************************************
***********/
void GenerateNetwork_Art(NET_Art* Net_Art)
{
INT i;

Net_Art->F1 = (LAYER_Art*) malloc(sizeof(LAYER_Art));
Net_Art->F2 = (LAYER_Art*) malloc(sizeof(LAYER_Art));

Net_Art->F1->Units     = N_Art;
Net_Art->F1->Output    = (BOOL*)  calloc(N_Art, sizeof(BOOL));
Net_Art->F1->Weight    = (REAL**) calloc(N_Art, sizeof(REAL*));
Net_Art->F2->Units     = M_Art;
```

```
    Net_Art->F2->Output    = (BOOL*)  calloc(M_Art, sizeof(BOOL));
Net_Art->F2->Weight    = (REAL**) calloc(M_Art, sizeof(REAL*));
Net_Art->F2->Inhibited = (BOOL*)  calloc(M_Art, sizeof(BOOL));

for (i=0; i<N_Art; i++) {
Net_Art->F1->Weight[i] = (REAL*) calloc(M_Art, sizeof(REAL));
}
for (i=0; i<M_Art; i++) {
Net_Art->F2->Weight[i] = (REAL*) calloc(N_Art, sizeof(REAL));
}
}

void SetInput_Art(NET_Art* Net_Art, BOOL* Input)
{
INT  i;
REAL Activation;

for (i=0; i<Net_Art->F1->Units; i++) {
Activation = Input[i] / (1 + Net_Art->A1 * (Input[i] + Net_Art->B1) + Net_Art-
>C1);
Net_Art->F1->Output[i] = (Activation > 0);
}
}

void GetOutput_Art(NET_Art* Net_Art, BOOL* Output)
{
INT i;
```

```
    for (i=0; i<Net_Art->F2->Units; i++) {
Output[i] = Net_Art->F2->Output[i];
}
}
/*****************************************************************
************
Propaga sinais para camada F2
*****************************************************************
***********/
void PropagateToF2(NET_Art* Net_Art)
{
INT  i,j;
REAL Sum, MaxOut;

MaxOut = MIN_REAL;
Net_Art->Winner = NO_WINNER;

for (i=0; i<Net_Art->F2->Units; i++) {
if (NOT Net_Art->F2->Inhibited[i]) {
Sum = 0;
for (j=0; j<Net_Art->F1->Units; j++) {
      Sum += Net_Art->F2->Weight[i][j] * Net_Art->F1->Output[j];
}
if (Sum > MaxOut) {
      MaxOut = Sum;
      Net_Art->Winner = i;
}
}
```

```
    Net_Art->F2->Output[i] = FALSE;
}
if (Net_Art->Winner != NO_WINNER)
Net_Art->F2->Output[Net_Art->Winner] = TRUE;
}

/*****************************************************************
************
Propaga sinais para camada F1
*****************************************************************
***********/

void PropagateToF1(NET_Art* Net_Art, BOOL* Input)
{
INT  i;
REAL Sum, Activation;

for (i=0; i<Net_Art->F1->Units; i++) {
Sum      =      Net_Art->F1->Weight[i][Net_Art->Winner]      *      Net_Art->F2-
>Output[Net_Art->Winner];
Activation = (Input[i] + Net_Art->D1 * Sum - Net_Art->B1) /
                        (1  +  Net_Art->A1  *  (Input[i]  +  Net_Art->D1  *  Sum)  +
Net_Art->C1);
Net_Art->F1->Output[i] = (Activation > 0);
}
}
```

```
    /*****************************************************************
***************
Ajustando Pesos
*****************************************************************
***********/
REAL Magnitude(NET_Art* Net_Art, BOOL* Input)
{
INT  i;

REAL Magnitude;

Magnitude = 0;
for (i=0; i<Net_Art->F1->Units; i++) {
Magnitude += Input[i];
}
return Magnitude;
}

void AdjustWeights_Art(NET_Art* Net_Art)
{
INT  i;
REAL MagnitudeInput_;

for (i=0; i<Net_Art->F1->Units; i++) {
if (Net_Art->F1->Output[i]) {
MagnitudeInput_ = Magnitude(Net_Art, Net_Art->F1->Output);
Net_Art->F1->Weight[i][Net_Art->Winner] = 1;
```

```
    Net_Art->F2->Weight[Net_Art->Winner][i] = Net_Art->L / (Net_Art->L - 1 +
MagnitudeInput_);
}
else {
Net_Art->F1->Weight[i][Net_Art->Winner] = 0;
Net_Art->F2->Weight[Net_Art->Winner][i] = 0;
}
}
}
/*****************************************************************
************
S imulando a rede ART
*****************************************************************
***********/
void SimulateNet_Art(NET_Art* Net_Art, BOOL* Input, BOOL* Output)
{
INT  i;
BOOL Resonance, Exhausted;
REAL MagnitudeInput, MagnitudeInput_;

WriteInput_Art(Input);
for (i=0; i<Net_Art->F2->Units; i++) {
Net_Art->F2->Inhibited[i] = FALSE;
}
Resonance = FALSE;
Exhausted = FALSE;
do {
SetInput_Art(Net_Art, Input);
```

```
      PropagateToF2(Net_Art);
GetOutput_Art(Net_Art, Output);
if (Net_Art->Winner != NO_WINNER) {
PropagateToF1(Net_Art, Input);
MagnitudeInput = Magnitude(Net_Art, Input);
MagnitudeInput_ = Magnitude(Net_Art, Net_Art->F1->Output);
if ((MagnitudeInput_ / MagnitudeInput) < Net_Art->Rho)
Net_Art->F2->Inhibited[Net_Art->Winner] = TRUE;
else
Resonance = TRUE;
}
else Exhausted = TRUE;
} while (NOT (Resonance OR Exhausted));
if (Resonance)
AdjustWeights_Art(Net_Art);
WriteOutput_Art(Net_Art);
}

void MostraAmbiente(void)
{
for(int i=0;i<25;i++)
{  for(int j=0; j<25; j++)
{
if(Ambiente[i][j]==true)
{
        gotoxy(j+1,i+1);
        printf("X");
}
```

```
    }
}
gotoxy(Agent.x,Agent.y);
printf("A");
}


/*****************************************************************
************
Procedimento para interpretar posição dos obstáculos
em relação a posição do agente no mundo
false para ausência, true para presença

*****************************************************************
**********/

int InterpretaAmbiente(void)
{
                //esquerda                                                cima
direita                         abaixo
if((Ambiente[(Agent.y-1)][(Agent.x-1)-1] == false) && (Ambiente[(Agent.y-1)-1]
[(Agent.x-1)] == false) && (Ambiente[(Agent.y-1)][(Agent.x-1)+1] == false) &&
(Ambiente[(Agent.y-1)+1][(Agent.x-1)] == false))
return 0;
else if((Ambiente[(Agent.y-1)][(Agent.x-1)-1] == false)
&& (Ambiente[(Agent.y-1)-1][(Agent.x-1)] == false)
&& (Ambiente[(Agent.y-1)][(Agent.x-1)+1] == false)
&& (Ambiente[(Agent.y-1)+1][(Agent.x-1)] == true))
```

```
    return 1;
else if((Ambiente[(Agent.y-1)][(Agent.x-1)-1] == false)
&& (Ambiente[(Agent.y-1)-1][(Agent.x-1)] == false)
&& (Ambiente[(Agent.y-1)][(Agent.x-1)+1] == true)
&& (Ambiente[(Agent.y-1)+1][(Agent.x-1)] == false))
return 2;
else if((Ambiente[(Agent.y-1)][(Agent.x-1)-1] == false)
&& (Ambiente[(Agent.y-1)-1][(Agent.x-1)] == false)
&& (Ambiente[(Agent.y-1)][(Agent.x-1)+1] == true)
&& (Ambiente[(Agent.y-1)+1][(Agent.x-1)] == true))
return 3;
else if((Ambiente[(Agent.y-1)][(Agent.x-1)-1] == false)
&& (Ambiente[(Agent.y-1)-1][(Agent.x-1)] == true)
&& (Ambiente[(Agent.y-1)][(Agent.x-1)+1] == false)
&& (Ambiente[(Agent.y-1)+1][(Agent.x-1)] == false))
return 4;
else if((Ambiente[(Agent.y-1)][(Agent.x-1)-1] == false)
&& (Ambiente[(Agent.y-1)-1][(Agent.x-1)] == true)
&& (Ambiente[(Agent.y-1)][(Agent.x-1)+1] == false)
&& (Ambiente[(Agent.y-1)+1][(Agent.x-1)] == true))
return 5;
else if((Ambiente[(Agent.y-1)][(Agent.x-1)-1] == false)
&& (Ambiente[(Agent.y-1)-1][(Agent.x-1)] == true)
&& (Ambiente[(Agent.y-1)][(Agent.x-1)+1] == true)
&& (Ambiente[(Agent.y-1)+1][(Agent.x-1)] == false))
return 6;
else if((Ambiente[(Agent.y-1)][(Agent.x-1)-1] == false)
&& (Ambiente[(Agent.y-1)-1][(Agent.x-1)] == true)
```

```
    && (Ambiente[(Agent.y-1)][(Agent.x-1)+1] == true)
&& (Ambiente[(Agent.y-1)+1][(Agent.x-1)] == true))
return 7;
else if((Ambiente[(Agent.y-1)][(Agent.x-1)-1] == true)
&& (Ambiente[(Agent.y-1)-1][(Agent.x-1)] == false)
&& (Ambiente[(Agent.y-1)][(Agent.x-1)+1] == false)
&& (Ambiente[(Agent.y-1)+1][(Agent.x-1)] == false))
return 8;
else if((Ambiente[(Agent.y-1)][(Agent.x-1)-1] == true)
&& (Ambiente[(Agent.y-1)-1][(Agent.x-1)] == false)
&& (Ambiente[(Agent.y-1)][(Agent.x-1)+1] == false)
&& (Ambiente[(Agent.y-1)+1][(Agent.x-1)] == true))
return 9;
else if((Ambiente[(Agent.y-1)][(Agent.x-1)-1] == true)
&& (Ambiente[(Agent.y-1)-1][(Agent.x-1)] == false)
&& (Ambiente[(Agent.y-1)][(Agent.x-1)+1] == true)
&& (Ambiente[(Agent.y-1)+1][(Agent.x-1)] == false))
return 10;
else if((Ambiente[(Agent.y-1)][(Agent.x-1)-1] == true)
&& (Ambiente[(Agent.y-1)-1][(Agent.x-1)] == false)
&& (Ambiente[(Agent.y-1)][(Agent.x-1)+1] == true)
&& (Ambiente[(Agent.y-1)+1][(Agent.x-1)] == true))
return 11;
else if((Ambiente[(Agent.y-1)][(Agent.x-1)-1] == true)
&& (Ambiente[(Agent.y-1)-1][(Agent.x-1)] == true)
&& (Ambiente[(Agent.y-1)][(Agent.x-1)+1] == false)
&& (Ambiente[(Agent.y-1)+1][(Agent.x-1)] == false))
return 12;
```

```
      else if((Ambiente[(Agent.y-1)][(Agent.x-1)-1] == true)
&& (Ambiente[(Agent.y-1)-1][(Agent.x-1)] == true)
&& (Ambiente[(Agent.y-1)][(Agent.x-1)+1] == false)
&& (Ambiente[(Agent.y-1)+1][(Agent.x-1)] == true))
return 13;
else if((Ambiente[(Agent.y-1)][(Agent.x-1)-1] == true)
&& (Ambiente[(Agent.y-1)-1][(Agent.x-1)] == true)
&& (Ambiente[(Agent.y-1)][(Agent.x-1)+1] == true)
&& (Ambiente[(Agent.y-1)+1][(Agent.x-1)] == false))
return 14;
else
return false;
}
/*-----------------------------------------[]
void SetInput(INT ID_Rede, REAL* Input)
[]-----------------------------------------*/
void SetInput(INT ID_Rede, REAL* Input)
{
INT i;

for (i=1; i <= Net[ID_Rede]->InputLayer->Units; i++) {
Net[ID_Rede]->InputLayer->Output[i] = Input[i-1];
}
}

/*-------------------------------------------------------[]
void PropagateLayer(NET* Net, LAYER* Lower, LAYER* Upper)
[]-------------------------------------------------------*/
```

```
    void PropagateLayer(NET* Net, LAYER* Lower, LAYER* Upper)
{
INT  i,j;
REAL Sum;

for (i=1; i<=Upper->Units; i++) {
Sum = 0;
for (j=0; j<=Lower->Units; j++) {
Sum += Upper->Weight[i][j] * Lower->Output[j];
}
/*Funcao de saida de cda neuronio: Sigmoidal*/
Upper->Output[i] = 1 / (1 + exp(-Net->Gain * Sum));

/*Funcao de saida de cda neuronio: Tangente Hiperbolica*/
//Upper->Output[i] = (1 - exp(-Net->Gain * Sum)) / (1 + exp(-Net->Gain *
Sum));
}

}


/*-----------------------------------------[]
void PropagateNet(INT ID_Rede)
[]-----------------------------------------*/
void PropagateNet(INT ID_Rede)
{
INT l;
```

```
      for (l=0; l < NUM_LAYERS-1; l++) {
PropagateLayer(Net[ID_Rede], Net[ID_Rede]->Layer[l], Net[ID_Rede]-
>Layer[l+1]);
}
}
/*-----------------------------------------[]
void GetOutput(INT ID_Rede, REAL* Output)
[]-----------------------------------------*/
void GetOutput(INT ID_Rede, REAL* Output)
{
INT i;

for (i=1; i <= Net[ID_Rede]->OutputLayer->Units; i++) {
Output[i-1] = Net[ID_Rede]->OutputLayer->Output[i];
}
}

/*****************************************************************
************
Implementação da Punição
Definição da conexão, linha e coluna da matriz de pesos,
para escolha aleatória do peso a ser punido (modificado)

*****************************************************************
***********/

void Punicao (INT index)
{
```

```
    int camada, linha, coluna;

camada = RandomEqualINT(1, NUM_LAYERS-1);
linha =  RandomEqualINT(1, Net[index]->Layer[camada]->Units);
coluna = RandomEqualINT(0, Net[index]->Layer[camada-1]->Units);

Net[index]->Layer[camada]->Weight[linha][coluna] = RandomEqualREAL(-1, 1);
}

/*****************************************************************
************
Treinamento em tempo real

*****************************************************************
***********/

void TrainingOnLine(int index)
{
REAL Output[M];
INT SaidaRedeInterpretada[M], Um;
INT Recompensa = false, i, j, Rede_a_punir, Coluna, Linha;

while(!Recompensa)
{     Um=0;
 SetInput(index, Situacoes[index]);
 PropagateNet(index);
 GetOutput(index, Output);
```

```
/*Limiar para a saida da rede valores acima de 0.5
  são interpretados como 1 e abaixo deste limiar como 0*/                    */

if(Output[0] < 0.5) SaidaRedeInterpretada[0] = 0;
else SaidaRedeInterpretada[0] = 1;

if(Output[1] < 0.5) SaidaRedeInterpretada[1] = 0;
else SaidaRedeInterpretada[1] = 1;

if(Output[2] < 0.5) SaidaRedeInterpretada[2] = 0;
else SaidaRedeInterpretada[2] = 1;

if(Output[3] < 0.5) SaidaRedeInterpretada[3] = 0;
else SaidaRedeInterpretada[3] = 1;

//Contando a qtidade de 1's

for(int i=0;i<4;i++)
      if(SaidaRedeInterpretada[i]==1)Um++;

/*Punicao a curto prazo - o agente só pode ter uma direção de saída
caso contrário será punido - terá seus pesos modificados*/
if(Um > 1 || Um ==0)Punicao(index);

/*Punicao a longo prazo*/

else if(Iteracoes >= Iteracoes_Maximas)
```

```
      {
if(Repeticao_Comportamento_Agente())
{
//Verifica quais as redes envolvidas na repeticao
for(i=0; i<Iteracoes_Maximas; i++)
{
Rede_a_punir = Guarda_Posicoes_Agente[i][2];
for(j=0; j<Iteracoes_Maximas;j++)
{
if(Rede_a_punir == Guarda_Posicoes_Agente[j][2])
Guarda_Posicoes_Agente[j][2]  = -1;
}
if(Rede_a_punir != -1)
{
gotoxy(30,17);
printf("Punicao Longo Prazo: MLP %i <TECLE ALGO> ", Rede_a_punir);
Sleep(500);
//getch();
gotoxy(30,17);
printf("                                        ");
//Punicao
RandomWeights(Rede_a_punir);
}
}
}
Iteracoes = 0;
 }
//Outros casos de punicao a curto prazo
```

```
    else
{
//Ver para onde o agente vai se mover
Coluna = Agent.x; Linha = Agent.y;
if (SaidaRedeInterpretada[0] == true)
Coluna = Agent.x - 1;
else if (SaidaRedeInterpretada[1] == true)
Linha = Agent.y - 1;
else if (SaidaRedeInterpretada[2] == true)
Coluna = Agent.x + 1;
else if (SaidaRedeInterpretada[3] == true)
Linha = Agent.y + 1;


/* punição de acordo com a impossibilidade de se mover
presença de obstáculos - chaveador (case) com as 15 possibilidades
de obstáculo */
switch (situacao_index)
{
case 0: //0000
Punicao(index);
//RandomWeights(index);
else Recompensa = true;
break;
case 1: //0001
if(SaidaRedeInterpretada[3] == 1 )Punicao(index);
else Recompensa = true;
break;
```

```
      case 2: //0010
if(SaidaRedeInterpretada[2]==1 )Punicao(index);
else Recompensa = true;
break;
case 3: //0011
if(SaidaRedeInterpretada[2]==1 || SaidaRedeInterpretada[3]==1
)Punicao(index);
else Recompensa = true;
break;
case 4: //0100
if(SaidaRedeInterpretada[1]==1)Punicao(index);
else Recompensa = true;
break;
case 5: //0101
if(SaidaRedeInterpretada[1]==1 || SaidaRedeInterpretada[3]==1
)Punicao(index);
else Recompensa = true;
break;
case 6: //0110
if(SaidaRedeInterpretada[1]==1 || SaidaRedeInterpretada[2]==1
)Punicao(index);
else Recompensa = true;
break;
case 7: //0111
if(SaidaRedeInterpretada[1]==1 || SaidaRedeInterpretada[2]==1 ||
SaidaRedeInterpretada[3]==1)Punicao(index);
else Recompensa = true;
break;
```

```
      case 8: //1000
if(SaidaRedeInterpretada[0]==1 )Punicao(index);
else Recompensa = true;
break;
case 9: //1001
if(SaidaRedeInterpretada[0]==1 || SaidaRedeInterpretada[3]==1
)Punicao(index);
else Recompensa = true;
break;
case 10: //1010
if(SaidaRedeInterpretada[0]==1 || SaidaRedeInterpretada[2]==1
)Punicao(index);
else Recompensa = true;
break;
case 11: //1011
if(SaidaRedeInterpretada[0] ==1 || SaidaRedeInterpretada[2]==1 ||
SaidaRedeInterpretada[3]==1)Punicao(index);
else Recompensa = true;
break;
case 12: //1100
if(SaidaRedeInterpretada[0]==1 || SaidaRedeInterpretada[1]==1
)Punicao(index);
else Recompensa = true;
break;
case 13: //1101
if(SaidaRedeInterpretada[0]==1 || SaidaRedeInterpretada[1] ==1 ||
SaidaRedeInterpretada[3]==1)Punicao(index);
else Recompensa = true;
```

```
      break;
case 14: //1110
if(SaidaRedeInterpretada[0]==1 || SaidaRedeInterpretada[1]==1 ||
SaidaRedeInterpretada[2]==1)Punicao(index);
else Recompensa = true;
break;
}//fim switch
}// fim else
}//fim while
ProxEstado.esq = SaidaRedeInterpretada[0];
ProxEstado.cima = SaidaRedeInterpretada[1];
ProxEstado.dir = SaidaRedeInterpretada[2];
ProxEstado.baixo = SaidaRedeInterpretada[3];

gotoxy(30,5);
printf("Saida da Rede MLP Interpretada: [%d %d %d
%d]",SaidaRedeInterpretada[0],
SaidaRedeInterpretada[1],SaidaRedeInterpretada[2],SaidaRedeInterpretada[3]);
//getch();
}

void MoverAgente(INT index_MLP)
{
//Limpar a posicao onde o Agente se encontra
gotoxy(Agent.x,Agent.y);
printf(" ");

//Ver para onde o agente vai se mover
```

```
      if  (ProxEstado.esq == true)
Agent.x -= 1;
else if (ProxEstado.cima == true)
Agent.y -= 1;
else if (ProxEstado.dir == true)
Agent.x += 1;
else if (ProxEstado.baixo == true)
Agent.y += 1;



gotoxy(Agent.x,Agent.y);
printf("A");
                                                                    //coluna
Guarda_Posicoes_Agente[Iteracoes][0] = Agent.x;
                                                                    //linha
Guarda_Posicoes_Agente[Iteracoes][1] = Agent.y;
                                                                    //indice
da rede MLP
Guarda_Posicoes_Agente[Iteracoes][2] = index_MLP;



Iteracoes++;
}



/*-------------------------------------------------------------------------[]
Rede ART aloca uma rede direta
ClasseSaida – índice da rede direta escolhida para resolver a situação encontrada
```

```
      Pelo agente
[]----------------------------------------------------------------------*/
void Treinamento(void)
{
NET_Art Net_Art;
INT n;

GenerateNetwork_Art(&Net_Art);
InitializeApplication_Art(&Net_Art);

while(true)
{
//Retorna a situacao do ambiente
situacao_index = InterpretaAmbiente();

if(situacao_index >= 0 && situacao_index <15)
{
SimulateNet_Art(&Net_Art, Input_Art[situacao_index], Output);
if(ClasseSaida >= 0 && ClasseSaida <15)
{
        TrainingOnLine(ClasseSaida);
        //RandomWeights(ClasseSaida);
        MoverAgente(ClasseSaida);
        Sleep(300);
}
else
{
        gotoxy(30,16);
```

```
            printf("Opcao da Rede MLP invalida");  getch();
        gotoxy(30,16);
        printf("                          ");
}

}
else
{   gotoxy(30,15);
 printf("Indice da Situacao invalida"); getch();
 gotoxy(30,15);
 printf("                          ");
}
}
gotoxy(30,23);
getch();
}

/*-----------------------------------------------------------------[]
Funcoes da rede neural direta
[]-----------------------------------------------------------------*/
void InitializeRandoms()
{
int semente = 1975;
//       printf ("\nSemente para inicializar o gerador de numeros pseudo-aleatorios:
");
//       scanf ("%d", &semente);
srand(semente);
}
```

```
    /*----------------------------------------------------------------[]
Funcoes de inicializacao da Rede Neural
[]----------------------------------------------------------------*/
void GenerateNetwork(INT ID_Rede)
{
INT l,i;

Net[ID_Rede] = (NET*) malloc(sizeof(NET));
// Net[ID_Rede] = *Net;

Net[ID_Rede]->Layer = (LAYER**) calloc(NUM_LAYERS, sizeof(LAYER*));

for (l=0; l<NUM_LAYERS; l++) {
Net[ID_Rede]->Layer[l] = (LAYER*) malloc(sizeof(LAYER));

Net[ID_Rede]->Layer[l]->Units      = Units[ID_Rede][l];
Net[ID_Rede]->Layer[l]->Output          = (REAL*)    calloc(Units[ID_Rede][l]+1,
sizeof(REAL));
Net[ID_Rede]->Layer[l]->Error           = (REAL*)    calloc(Units[ID_Rede][l]+1,
sizeof(REAL));
Net[ID_Rede]->Layer[l]->Weight          = (REAL**)  calloc(Units[ID_Rede][l]+1,
sizeof(REAL*));
Net[ID_Rede]->Layer[l]->WeightSave   =  (REAL**)   calloc(Units[ID_Rede][l]+1,
sizeof(REAL*));
Net[ID_Rede]->Layer[l]->dWeight        =  (REAL**)  calloc(Units[ID_Rede][l]+1,
sizeof(REAL*));
Net[ID_Rede]->Layer[l]->Output[0]  = BIAS;
```

```
      if (l != 0) {
for (i=1; i<=Units[ID_Rede][l]; i++) {
Net[ID_Rede]->Layer[l]->Weight[i]       = (REAL*) calloc(Units[ID_Rede][l-1]+1,
sizeof(REAL));
Net[ID_Rede]->Layer[l]->WeightSave[i] = (REAL*) calloc(Units[ID_Rede][l-1]+1,
sizeof(REAL));
Net[ID_Rede]->Layer[l]->dWeight[i]     = (REAL*) calloc(Units[ID_Rede][l-1]+1,
sizeof(REAL));
}
}
}
Net[ID_Rede]->InputLayer  = Net[ID_Rede]->Layer[0];
Net[ID_Rede]->OutputLayer = Net[ID_Rede]->Layer[NUM_LAYERS - 1];
Net[ID_Rede]->Alpha       = 0.4;    /*taxa de momentum*/
Net[ID_Rede]->Eta         = 0.5;  /*taxa de aprendizagem*/
Net[ID_Rede]->Gain        = 1;
}
/*----------------------------------------------------------------------[]
Libera Memoria utilizada pela RNA de indice ID_Rede
[]----------------------------------------------------------------------*/
void LiberaMemoriaNetwork(NET *Rede, INT ID_Rede)
{
for (int l=0; l<NUM_LAYERS; l++)
{ free(Rede->Layer[l]->Output);
free(Rede->Layer[l]->Error);

if (l != 0) {
for (int i=1; i<=Units[ID_Rede][l]; i++) {
```

```
      free(Rede->Layer[l]->Weight[i]);
free(Rede->Layer[l]->WeightSave[i]);
free(Rede->Layer[l]->dWeight[i]);
}
}
free(Rede->Layer[l]->Weight);
free(Rede->Layer[l]->WeightSave);
free(Rede->Layer[l]->dWeight);

free(Rede->Layer[l]);
}
free(Rede->Layer);

free(Rede);
Rede = NULL;
}
/*----------------------------------------[]
 MAIN
[]----------------------------------------*/
void   main  ( void )
{
BOOL Stop;
REAL MinTestError;
INT  n = 0, i;

InitializeRandoms();
/*Gerando as 15 redes neurais*/
while( n < 15 )
```

```
    { Units[n][0] = N;
Units[n][1] = 4;
Units[n][2] = M;
GenerateNetwork(n);
RandomWeights(n);
n++;
}
clrscr();
//Posicao inicial do agente
Agent.x = 2;
Agent.y = 24;

MostraAmbiente();
Treinamento();
}
```